

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

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 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"/>
sequences	<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 ingenjörstudenter 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 och i kursrepot 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

Framstegsprotokoll	3
Förord	5
I Om kursen	3
Kursens arkitektur	5
Anvisningar	9
Samarbetsgrupper	9
Föreläsningar	9
Övningar	9
Laborationer	9
Resurstider	9
Kontrollskrivning	9
Tentamen	9
Hur bidra till kursmaterialet?	11
II Moduler	15
1 Introduktion	17
1.1 Vad är programmering?	18
1.2 Vad är en kompilator?	18
1.3 Vad består ett program av?	19
1.4 Exempel på programmeringsspråk	19
1.5 Varför Scala + Java som förstaspråk?	20
1.6 Hello world	20
1.7 Utvecklingscykeln	21
1.8 Utvecklingsverktyg	21
1.9 Övning: expressions	22
1.9.1 Grunduppgifter	22
1.9.2 Extrauppgifter: öva mer på grunderna	29
1.9.3 Fördjupningsuppgifter: avancerad nivå	30
1.10 Laboration: kojo	32
1.10.1 Obligatoriska uppgifter	32

1.10.2	Frivilliga extrauppgifter	38
2	Kodstrukturer	41
2.1	Övning: programs	42
2.1.1	Grunduppgifter	42
2.1.2	Extrauppgifter: öva mer på grunderna	51
2.1.3	Fördjupningsuppgifter: avancerad nivå	51
3	Funktioner, Objekt	53
3.1	Övning: functions	54
3.1.1	Grunduppgifter	54
3.1.2	Extrauppgifter: öva mer på grunderna	62
3.1.3	Fördjupningsuppgifter: avancerad nivå	63
3.2	Laboration: simplewindow	65
3.2.1	Obligatoriska uppgifter	65
3.2.2	Frivilliga extrauppgifter	65
4	Datastrukturer	67
4.0.3	Att göra denna vecka	68
4.1	Denna vecka: Fatta datastrukturer	68
4.1.1	Olika sorters datastrukturer	68
4.2	Olika sätt att skapa datastrukturer	68
4.2.1	Tupler	68
4.3	Vad är en tupel?	68
4.4	Övning: data	69
4.4.1	Grunduppgifter	69
4.4.2	Extrauppgifter: öva mer på grunderna	82
4.4.3	Fördjupningsuppgifter: avancerad nivå	82
4.5	Laboration: textfiles	86
4.5.1	Obligatoriska uppgifter	86
4.5.2	Frivilliga extrauppgifter	86
5	Sekvensalgoritmer	87
5.1	Vad är en sekvensalgoritm?	88
5.2	Några indexerbara samlingar	88
5.3	Algoritm: SEQ-COPY	88
5.4	Övning: sequences	89
5.4.1	Grunduppgifter	89
5.4.2	Extrauppgifter: öva mer på grunderna	93
5.4.3	Fördjupningsuppgifter: avancerad nivå	93
5.5	Laboration: cardgame	94
5.5.1	Obligatoriska uppgifter	94
5.5.2	Frivilliga extrauppgifter	94

6	Klasser, Likhet	95
6.1	Övning: classes	96
6.1.1	Grunduppgifter	96
6.1.2	Extrauppgifter: öva mer på grunderna	96
6.1.3	Fördjupningsuppgifter: avancerad nivå	96
6.2	Laboration: shapes	97
6.2.1	Obligatoriska uppgifter	97
6.2.2	Frivilliga extrauppgifter	97
7	Arv, Gränssnitt	99
7.1	Övning: traits	100
7.1.1	Grunduppgifter	100
7.1.2	Extrauppgifter: öva mer på grunderna	100
7.1.3	Fördjupningsuppgifter: avancerad nivå	100
7.2	Laboration: turtlerace-team	101
7.2.1	Obligatoriska uppgifter	101
7.2.2	Frivilliga extrauppgifter	101
8	Mönster, Undantag	103
8.1	Övning: matching	104
8.1.1	Grunduppgifter	104
8.1.2	Extrauppgifter: öva mer på grunderna	104
8.1.3	Fördjupningsuppgifter: avancerad nivå	104
8.2	Laboration: chords-team	105
8.2.1	Obligatoriska uppgifter	105
8.2.2	Frivilliga extrauppgifter	105
9	Matriser, Typparametrar	107
9.1	Övning: matrices	108
9.1.1	Grunduppgifter	108
9.1.2	Extrauppgifter: öva mer på grunderna	108
9.1.3	Fördjupningsuppgifter: avancerad nivå	108
9.2	Laboration: maze	109
9.2.1	Obligatoriska uppgifter	109
9.2.2	Frivilliga extrauppgifter	109
10	Sökning, Sortering	111
10.1	Övning: sorting	112
10.1.1	Grunduppgifter	112
10.1.2	Extrauppgifter: öva mer på grunderna	112
10.1.3	Fördjupningsuppgifter: avancerad nivå	112
10.2	Laboration: surveydata-team	113
10.2.1	Obligatoriska uppgifter	113
10.2.2	Frivilliga extrauppgifter	113

11 Scala och Java	115
11.1 Övning: scalajava	116
11.1.1 Grunduppgifter	116
11.1.2 Extrauppgifter: öva mer på grunderna	116
11.1.3 Fördjupningsuppgifter: avancerad nivå	116
11.2 Laboration: scalajava-team	117
11.2.1 Obligatoriska uppgifter	117
11.2.2 Frivilliga extrauppgifter	117
12 Trådar, Web ???, Android ???	119
12.1 Övning: threads	120
12.1.1 Grunduppgifter	120
12.1.2 Extrauppgifter: öva mer på grunderna	120
12.1.3 Fördjupningsuppgifter: avancerad nivå	120
12.2 Laboration: life	121
12.2.1 Obligatoriska uppgifter	121
12.2.2 Frivilliga extrauppgifter	121
13 Design	123
14 Tentaträning	125
III Appendix	127
A Virtuellt maskin	129
A.1 Vad är en virtuellt maskin?	129
A.2 Installera kursens vm	129
A.3 Vad innehåller kursens vm?	130
B Terminalfönster och kommandoskal	131
B.1 Vad är ett terminalfönster?	131
B.2 Några viktiga terminalkommando	131
C Editera	133
C.1 Vad är en editor?	133
C.2 Välj editor	133
D Kompilera och exekvera	135
D.1 Vad är en kompilator?	135
D.2 Java JDK	135
D.2.1 Installera Java JDK	135
D.3 Scala	135
D.3.1 Installera Scala-kompilatorn	135
D.4 Read-Evaluate-Print-Loop (REPL)	135
D.4.1 Scala REPL	135

E Dokumentation	137
E.1 Vad gör ett dokumentationsverktyg?	137
E.2 scaladoc	137
E.3 javadoc	137
F Integrerad utvecklingsmiljö	139
F.1 Vad är en IDE?	139
F.2 Kojo	139
F.2.1 Installera Kojo	139
F.2.2 Använda Kojo	139
F.3 Eclipse och ScalaIDE	139
F.3.1 Installera Eclipse och ScalaIDE	139
F.3.2 Använda Eclipse och ScalaIDE	139
G Byggverktyg	141
G.1 Vad gör ett byggverktyg?	141
G.2 Byggverktyget sbt	141
G.2.1 Installera sbt	141
G.2.2 Använda sbt	141
H Versionshantering och kodlagring	143
H.1 Vad är versionshantering?	143
H.2 Versionshanteringsverktyget git	143
H.2.1 Installera git	143
H.2.2 Använda git	143
H.3 Vad är nyttan med en kodlagringsplats?	143
H.4 Kodlagringsplatsen GitHub	143
H.4.1 Installera klienten för GitHub	143
H.4.2 Använda GitHub	143
H.5 Kodlagringsplatsen Atlassian BitBucket	143
H.5.1 Installera SourceTree	143
H.5.2 Använda SourceTree	143
I Nyckelord	145
I.1 Vad är ett nyckelord ord?	145
I.2 Nyckelord i Scala	145
I.3 Nyckelord i Java	145
J Lösningsförslag till övningar	147
J.1 expressions	148
J.1.1 Grunduppgifter	148
J.1.2 Extrauppgifter: öva mer på grunderna	148
J.1.3 Fördjupningsuppgifter: avancerad nivå	148
J.2 programs	149
J.2.1 Grunduppgifter	149
J.2.2 Extrauppgifter: öva mer på grunderna	149
J.2.3 Fördjupningsuppgifter: avancerad nivå	149

J.3	functions	150
J.3.1	Grunduppgifter	150
J.3.2	Extrauppgifter: öva mer på grunderna	150
J.3.3	Fördjupningsuppgifter: avancerad nivå	150
J.4	data	151
J.4.1	Grunduppgifter	151
J.4.2	Extrauppgifter: öva mer på grunderna	151
J.4.3	Fördjupningsuppgifter: avancerad nivå	151
J.5	sequences	152
J.5.1	Grunduppgifter	152
J.5.2	Extrauppgifter: öva mer på grunderna	152
J.5.3	Fördjupningsuppgifter: avancerad nivå	152
J.6	classes	153
J.6.1	Grunduppgifter	153
J.6.2	Extrauppgifter: öva mer på grunderna	153
J.6.3	Fördjupningsuppgifter: avancerad nivå	153
J.7	traits	154
J.7.1	Grunduppgifter	154
J.7.2	Extrauppgifter: öva mer på grunderna	154
J.7.3	Fördjupningsuppgifter: avancerad nivå	154
J.8	matching	155
J.8.1	Grunduppgifter	155
J.8.2	Extrauppgifter: öva mer på grunderna	155
J.8.3	Fördjupningsuppgifter: avancerad nivå	155
J.9	matrices	156
J.9.1	Grunduppgifter	156
J.9.2	Extrauppgifter: öva mer på grunderna	156
J.9.3	Fördjupningsuppgifter: avancerad nivå	156
J.10	sorting	157
J.10.1	Grunduppgifter	157
J.10.2	Extrauppgifter: öva mer på grunderna	157
J.10.3	Fördjupningsuppgifter: avancerad nivå	157
J.11	scalajava	158
J.11.1	Grunduppgifter	158
J.11.2	Extrauppgifter: öva mer på grunderna	158
J.11.3	Fördjupningsuppgifter: avancerad nivå	158
J.12	threads	159
J.12.1	Grunduppgifter	159
J.12.2	Extrauppgifter: öva mer på grunderna	159
J.12.3	Fördjupningsuppgifter: avancerad nivå	159

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	Sekvensalgoritmer	sequences	cardgame
W06	Klasser, Likhet	classes	shapes
W07	Arv, Gränssnitt	traits	turtlerace-team
KS	KONTROLLSKRIVN.	–	–
W08	Mönster, Undantag	matching	chords-team
W09	Matriser, Typparametrar	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 övningarna och laborationen i en modul är du redo att gå vidare till nästa 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

Anvisningar

Samarbetsgrupper

Samarbetskontrakt

Föreläsningar

Övningar

Laborationer

Resurstider

Kontrollskrivning

Tentamen

Hur bidra till kursmaterialet?

Bidrag är varmt välkomna!

Ett av huvudsyftena med att göra detta kursmaterial fritt och öppet är att möjliggöra bidrag från alla som är intresserade. Speciellt välkommet är bidrag från studenter som vill vara delaktiga i att utveckla undervisningen.

Instruktioner

Vad behöver jag för att kunna bidra?

Om du hittar ett problem, t.ex. ett enkelt stavfel, eller har något mer omfattande som borde förbättras, men ännu inte känner till eller har tillgång till de verktyg som beskrivs nedan och som behövs för att göra bidrag, kontakta då någon som redan bidragit till materialet, så att någon annan kan implementera ditt förslag.

Innan du själv kan implementera ändringar direkt i materialet, behöver du känna till, och ha tillgång till, ett eller flera av följande verktyg (beroende på vad ändringen gäller):

- Latex: en.wikibooks.org/wiki/LaTeX
- Scala: en.wikipedia.org/wiki/Scala_%28programming_language%29
- git: https://en.wikipedia.org/wiki/Git_%28software%29
- GitHub: en.wikipedia.org/wiki/Github
- sbt: en.wikipedia.org/wiki/SBT_%28software%29

Läs mer om hur du bidrar här:

github.com/lunduniversity/introprog#how-to-contribute-to-this-repo

Svenska eller engelska?

Vi blandar engelska och svenska enligt följande principer:

- Publika diskussioner, t.ex. i issues och pull requests på GitHub, sker på engelska. I en framtid kan delar av materialet komma att översättas till engelska och då är det bra om även icke-engelskspråkiga kan förstå vad som har hänt. Alla ändringshändelser sparas och man kan söka och gå tillbaka i historiken.

- Kompendiet finns för närvarande bara på svenska eftersom kursen initialt endast ges för svenskspråkiga studenter, men texten ska hjälpa läsaren att tillgodogöra sig motsvarande engelsk terminologi. Skriv därför motsvarande engelska begrepp (eng. *concept*) i parentes med hjälp av latex-kommandot `\Eng{concept}`.
- På övningar och föreläsningar är svenska variabelnamn ok. Svenska kan användas för att hjälpa den som håller på att lära sig att skilja på ord som vi själv hittar på och ord som finns i programmeringsspråket. Detta signalerar också att när man lär sig och experimenterar kan man hitta på tokroliga namn och använda svenska hur mycket man vill. Man lär sig genom att prova!
- Kod i labbar ska vara på engelska. Detta signalerar att när man kodar för att det ska bli något bestående, då kodar man på engelska.

Exempel

Som exempel på hur det går till i ett typiskt öppen-källkodsprojekt, beskrivs nedan vad som hände i ett verkligt fall: en dokumentationsuppdatering av Scala-dokumentationen efter att ett fel upptäckts. Detta exempelfall är ett typiskt scenario som illustrerar hur det kan gå till, och vad man kan behöva tänka på. Exemplet ger också länkar till och inblick i ett riktigt stort projekt med öppen källkod.

Scenario: att göra ett bidrag vid upptäckt av problem

”Jag fick till min stora glädje denna *Pull Request* (PR) accepterad till dokumentationssajten för Scala. Man kan se mitt bidrag här:

github.com/scala/scala.github.com/commit/7da81868ba4d74b87fe0b1

Att börja med att bidra till dokumentation är ofta en bra väg att komma in i ett open source-projekt, då det är en god chans att hjälpa till utan att det behöver kräva djup kompetens om koden i repot. Jag beskriver nedan vad som hände steg för steg då jag fick en riktig PR accepterad, som ett typiskt exempel på hur det ofta fungerar.

1. Jag tyckte dokumentationen för metoden `lengthCompare` på indexerbara samlingar på scala-lang.org/documentation var förvirrande. När jag provade i REPL blev det uppenbart att något var fel: antingen så var dokumentationen fel eller så funkade inte metoden som den skulle. Ojoj, kanske har jag upptäckt ett nytt fel? En chans att bidra!
2. Först sökte jag noga bland alla issues som ligger under fliken 'issues' på GitHub för att se om någon redan hittat detta problem. Om så vore fallet hade jag kunnat kommentera en sådan issue och skriva något till stöd för att den behöver fixas, eller allra helst att erbjuda mig att försöka fixa den. Men jag hittade ingen issue om detta...

3. Jag skapade därför ett nytt ärende genom att klicka på knappen *New issue* i webbgränssnittet på GitHub och här syns resultatet:

<https://github.com/scala/scala.github.com/issues/515#>

Jag tänkte noga på hur jag skulle formulera mig:

- Titlen på issuen är extra viktig: den ska sammanfatta på en enda rad vad det hela rör sig om så att läsaren av rubriken förstår vad problemet är.
 - Jag jobbade sedan med att skriva en tydlig och detaljerad beskrivning av problemet och angav exakt vilken version det gällde. Det är bra att klistra in exempel från Scala REPL och andra testfallskörningar med indata och utdata om relevant. Det är viktigt att problemet går att hitta och återskapa av andra, därför behövs information om vilken version det gäller och ett minimalt testfall som renodlar problemet.
 - Det är bra att ställa frågor och komma med förslag för att öppna en diskussion om ärendet. Jag frågade speciellt om detta var ett dokumentationsproblem eller en bugg i koden.
 - OBS! Man ska inte öppna en issue innan man först kollat noga att det verkligen är något som bör åtgärdas och att det inte är en dubblett eller överlapp med andra issues: varje gång man öppnar ett ärende kommer det att generera arbete för andra även om issuen inte ens till slut åtgärdas...
 - Om det är ett mer öppet, allmänt förslag, en förbättring eller en helt ny feature kan man också skapa en issue (det måste alltså inte vara en renodlad bugg). Är man osäker på om ärendet är relevant, är det bra att diskutera det i gemenskapens mejlforum först.
4. Jag fick snabbt kommentarer på min issue, vilket är kännetecknande för en väl fungerande community med alerta maintainers. Och när jag fick uppmuntran att bidra, så erbjöd jag mig att implementera förbättringen. Tänk på att alltid skriva i en saklig, kortfattad och trevlig ton!
5. Nästa steg är att "forka" repot på GitHub genom att helt enkelt klicka på *Fork* i webbgränssnittet. Jag fick då en egen kopia av repot under min egen användare på GitHub, där jag har rättigheter att ändra.
6. Därefter klonade jag repot till min lokala maskin med terminalkommandot `git clone https://...` (eller så kan man använda skrivbordsappen GitHub Desktop).
7. Sedan rättade jag problemet direkt i relevant fil i en editor på min dator, i detta fallet var filen i formatet Markdown (ett lättläst textformat som man kan generera html från):
raw.githubusercontent.com/scala/scala.github.com/master/overviews/collections/seqs.md
8. När jag fixat problemet gjorde jag `git add` på filen och sedan `git commit -m "välgenomtänkt commit msg"`. Jag tänkte efter noga innan jag skrev första raden i commit-meddelandet så att det skulle vara både kort och

kärnfullt. Men ändå glömde jag att inkludera issue-numret : (, se min kommentar till commiten, som jag tillfogade i efterhand, när jag till slut upptäckte min fadäs:

scala.github.com/commit/2624c305a8a6f24ea3398fe0fcbd0c72492bdd12#comments

9. Efter att jag gjort `git commit` så finns ändringen ännu så länge bara lokalt på min dator. Då gäller det att "pusha" till min fork på GitHub med `git push` (eller använda *Synch*-knappen i GitHub-desktop-appen).
10. Därefter skapade jag en PR genom att helt enkelt trycka på knappen *New pull request* på GitHub-sidan för min fork. Jag tänkte efter noga innan jag författade rubriken som beskriver denna PR. Hade denna ändring varit mer omfattande hade jag också behövt göra en detaljerad beskrivning av hur ändringen var implementerad för att underlätta granskningen av mitt förslag. Ni kan se denna (numera avslutade) PR här:
<https://github.com/scala/scala.github.com/pull/517>
11. När jag skapat en PR fick de som sköter repot ett automatiskt meddelande om denna nya PR och den efterföljande granskningsfasen inträdde. Den brukar sluta med att en eller flera andra personer kommenterar PR i webbgränssnittet med 'LGTM'. LGTM = "*Looks Good To Me*" och betyder ungefär "jag har kollat på detta nu och det verkar (vad jag kan bedöma) vara utmärkt och alltså redo för *merge*". Om det inte ser bra ut så förväntas granskaren föreslå vad som behöver förbättras i en saklig och trevlig ton.
12. När PR är granskad så kan en person, som har rättigheter att ändra, "merge" in PR på huvudgrenen, som ofta kallas *master*, i det centrala repot, som ofta kallas *upstream*.
13. Avslutningsvis kan issuen stängas av de ansvariga för repot. Issuen är nu markerad "Closed" och syns inte längre i listan med aktiva issues.

Puh! Sen var det klart :) "

Epilog: Om du i framtiden får chansen att göra fler bidrag är det viktigt att först uppdatera din fork mot upstream innan du gör några nya ändringar i din lokala kopia; annars är risken att din PR innehåller föråldrad information och därmed blir en merge onödigt krånglig. Detta kan man göra genom en knapp i GitHub Desktop eller genom att följa denna beskrivning: help.github.com/articles/syncing-a-fork/ Det är i allmänhet den som ändrars ansvar att se till att ändringar alltid sker i samklang med den mest aktuella versionen av upstream.

Del II

Moduler

Kapitel 1

Introduktion

Koncept du ska lära dig denna vecka:

- | | |
|---|--|
| <input type="checkbox"/> sekvens | <input type="checkbox"/> Short |
| <input type="checkbox"/> alternativ | <input type="checkbox"/> Double |
| <input type="checkbox"/> repetition | <input type="checkbox"/> Float |
| <input type="checkbox"/> abstraktion | <input type="checkbox"/> Byte |
| <input type="checkbox"/> programmeringsspråk | <input type="checkbox"/> Char |
| <input type="checkbox"/> programmeringsparadigmer | <input type="checkbox"/> String |
| <input type="checkbox"/> editera-kompilera-exekvera | <input type="checkbox"/> println |
| <input type="checkbox"/> datorns delar | <input type="checkbox"/> typen Unit |
| <input type="checkbox"/> virtuell maskin | <input type="checkbox"/> enhetsvärdet () |
| <input type="checkbox"/> REPL | <input type="checkbox"/> stränginterpolatorn s |
| <input type="checkbox"/> literal | <input type="checkbox"/> if |
| <input type="checkbox"/> värde | <input type="checkbox"/> else |
| <input type="checkbox"/> uttryck | <input type="checkbox"/> true |
| <input type="checkbox"/> variabel | <input type="checkbox"/> false |
| <input type="checkbox"/> typ | <input type="checkbox"/> MinValue |
| <input type="checkbox"/> tilldelning | <input type="checkbox"/> MaxValue |
| <input type="checkbox"/> namn | <input type="checkbox"/> aritmetik |
| <input type="checkbox"/> val | <input type="checkbox"/> slumpstal |
| <input type="checkbox"/> var | <input type="checkbox"/> math.random |
| <input type="checkbox"/> def | <input type="checkbox"/> logiska uttryck |
| <input type="checkbox"/> inbyggda typer | <input type="checkbox"/> de Morgans lagar |
| <input type="checkbox"/> Int | <input type="checkbox"/> while-sats |
| <input type="checkbox"/> Long | <input type="checkbox"/> 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!



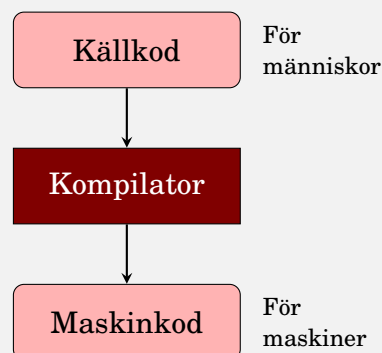
- sv.wikipedia.org/wiki/Programmering
- en.wikipedia.org/wiki/Computer_programming
- kartor.lund.se/wiki/lundanamn/index.php/Ada_Lovelace-parken

1.2 Vad är en kompilator?



Grace Hopper uppfann första kompilatorn 1952.

en.wikipedia.org/wiki/Grace_Hopper



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

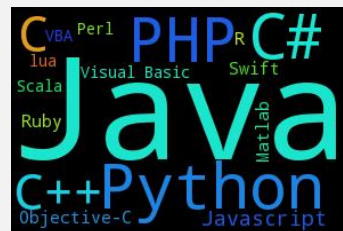
Det finns massor med olika språk och det kommer ständigt nya.

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 slumptal i olika intervall.

Förberedelser

- ☐ Studera teorin i kapitel 1.
- ☐ Du behöver en dator med Scala installerad, se appendix D.

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> res0 * 42
```


Uppgift 2. Vad är en *literal*?

[en.wikipedia.org/wiki/Literal_\(computer_programming\)](https://en.wikipedia.org/wiki/Literal_(computer_programming))

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, vad som är största och minsta värde, till exempel Int.MaxValue etc.

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

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

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

-
- a) Undersök genom att trycka på Tab-tangenten, vilka funktioner som finns i Math och math. Vad heter konstanten π 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 pakobjektet 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. Booelska 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 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. *Tilldelningsoperatorer.* Man kan förkorta en tilldelningssats som förändrar en variabel, t.ex. `x = x + 1`, genom att använda så kallade tilldelningsoperatorer och skriva `x += 1` som betyder samma sak. 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 + 42
3 scala> a += 10
4 scala> b -= 10
5 scala> a *= 2
6 scala> b /= 2
```

Uppgift 17. Stränginterpolatorn s. Man behöver ofta skapa strängar som innehåller variabelvärden. Med ett `s` framför en strängliteral får man hjälp av kompilatorn att infoga variabelvärden i en sträng. Variablernas namn ska föregås med ett `$`-tecken, t.ex. `s"Hej $namn"`.

```
1 scala> val f = "Kim"
2 scala> val e = "Robinson"
3 scala> val tot = f.size + e.size
4 scala> println(s"Namnet '$f $e' har $tot bokstäver.")
```

- a) Vad skrivs ut ovan?
- b) Skapa följande utskrifter med hjälp av stränginterpolatorn `s` och lämpliga variabler.

```
1 Namnet 'Kim' har 3 bokstäver.
2 Namnet 'Robinson' har 9 bokstäver.
```

Uppgift 18. if-sats. För varje rad nedan; förklara vad som händer.

```
1 scala> if (true) println("sant") else println("falskt")
2 scala> if (false) println("sant") else println("falskt")
3 scala> if (!true) println("sant") else println("falskt")
4 scala> if (!false) println("sant") else println("falskt")
5 scala> def kasta = if (math.random > 0.5) print(" krona") else print(" klave")
6 scala> kasta; kasta; kasta
```

Uppgift 19. if-uttryck. Deklarera följande variabler 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"`
- d) `if (true) grönsak else frukt`
- e) `if (false) grönsak else frukt`

Uppgift 20. for-sats.

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

```

1 scala> for (i <- 1 to 10) print(i + ", ")
2 scala> for (i <- 1 until 10) print(i + ", ")
3 scala> for (i <- 1 to 5) print((i * 2) + ", ")
4 scala> for (i <- 1 to 92 by 10) print(i + ", ")
5 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 21. Repetition med foreach.

a) Vad ger nedan satser för utskrifter?

```

1 scala> (9 to 19).foreach{i => print(i + ", ")}
2 scala> (1 until 20).foreach{i => print(i + ", ")}
3 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 22. while-sats.

a) Vad ger nedan satser för utskrifter?


```

1 scala> var i = 0
2 scala> while (i < 10) { println(i); i = i + 1 }
3 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 23. Slumptal. Undersök vad dokumentationen säger om funktionen `scala.math.random`:

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

- Vilken typ har värdet som returneras av funktionen `random`?
- Vilket är det minsta respektive största värde som kan returneras?
- Är `random` en *äkta* funktion (eng. *pure function*) i matematisk mening?
- 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.

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

Uppgift 24. *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 25. *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 26. Läs om modulatoräkning här en.wikipedia.org/wiki/Modulo_operation och undersök hur tecknet blir med olika tecken på divisor och dividend.

Uppgift 27. `Integer.toString`, `Integer.toHexString`

Uppgift 28. Typannoteringar.

Uppgift 29. `0x2a`

Uppgift 30. `i += 1; i *= 1; i /= 2`

Uppgift 31. `BigInt`, `BigDecimal`

Uppgift 32. Vad händer här?

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

Uppgift 33. Sök reda på dokumentationen i javadoc för klassen `java.lang.Math` i JDK 8. Tryck `Ctrl+F` i webbläsaren och sök efter förekomster av texten "overflow". Vad är "overflow"? Vilka metoder finns i `java.lang.Math` som hjälper dig att upptäcka om det blir overflow?

Uppgift 34. 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 35. 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.toString(Int.MaxValue + 1).size
```

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

```
1 + math.log(Long.MaxValue)/math.log(2)
```

Se även språkspecifikationen för Scala, kapitlet om heltalsliteraler:

<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?

Uppgift 36. Läs mer om stränginterpolatorer här:

docs.scala-lang.org/overviews/core/string-interpolation.html

Hur kan du använda f-interpolatorn för att göra följande utskrift i REPL? Byt ut `???` mot lämpliga tecken.

```
scala> val g: Double = 1 / 3.0
scala> val s: String = f"Gurkan är ??? meter lång"
scala> println(s)
Gurkan är 0.333 meter lång
```

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 F.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 F.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.

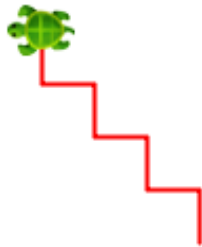
c) Prova satser i sekvens på flera rader, respektive på samma rad med semikolon emellan. Hur vill du gruppera dina satser så att de lätta för en människa att läsa?

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.
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.
- Kontrollera med hjälp av `math.hypot` och `println` vad det exakta svaret är. Skriv ner svaret med 3 decimalers noggrannhet.

h) Rita en triangel med sidan 300 längdenheter genom att ge lämpliga argument till fram och höger. Vinklar anges i 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 en repetition för abstrahera nedan sekvens, så att programmet blir kortare:

```
sudda

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

b) Sök på nätet efter "DRY principle programming" och beskriv med egna ord vad DRY betyder och varför det är en viktig princip. 📎

c) 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}
```

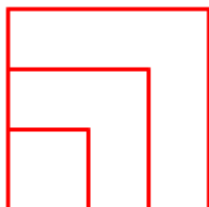


- d) Kör ditt program med den *gula* play-knappen. Studera hur anrop av proceduren *kvadrat* påverkar exekveringssekvensen av dina satser. Vid vilka punkter i programmet sker ett ”hopp” i sekvensen i stället för att efterföljande sats att exekveras? Använd lämpligt argument till *sakta* för att du ska hinna studera exekveringen.
- e) 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 abstraktionen *kvadrat*) 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.
- f) Skapa en abstraktion **def** *stapel* = ??? med din kod för att rita en stapel.
- g) Du ska nu generalisera din procedur så att den inte bara kan rita exakt 10 kvadrater i en stapel. Ge proceduren *stapel* en parameter *n* som styr hur många kvadrater som ritas.

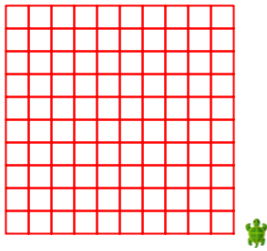
```
def kvadrat = ???
def stapel(n: Int) = ???

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

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



- i) 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)



- j) 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.
- k) 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 handledare. Vad har du gjort för att programmet ska vara lättläst?



Uppgift 4. Variabel.

- a) Skriv in nedan program *exakt* som det står med blanktecken, indragningar och radbrytningar. Kör programmet och förklara vad som händer.

```
def gurka(x: Double,
          y: Double, namn: String,
          typ: String,
          värde:String) = {
  val bredd = 15
  val h = 30
  hoppaTill(x,y)
  norr
  skriv(namn+": "+typ)
  hoppaTill(x+bredd*(namn.size+typ.size),y)
  skriv(värde); söder; fram(h); vänster
  fram(bredd * värde.size); vänster
  fram(h); vänster
  fram(bredd * värde.size); vänster
}

sudda; färg(svart)
val s = 130
val h = 40
var x = 42; gurka(10, s-h*0, "x","Int", x.toString)
var y = x; gurka(10, s-h*1, "y","Int", y.toString)
x = x + 1; gurka(10, s-h*2, "x","Int", x.toString)
          gurka(10, s-h*3, "y","Int", y.toString)
osynlig
```

- b) Skriv ner namnet på alla variabler som förekommer i programmet ovan.





- c) Vilka av dessa variabler är lokala?
- d) Vilka av dessa variabler kan förändras?
- e) Föreslå tre förändringar av programmet ovan (till exempel namnbyten) som gör att det blir lättare att läsa och förstå.
- f) Gör sök-ersätt av gurka till ett bättre namn. *Tips:* undersök kontextmenyn i editorn i Kojo genom att högerklicka i editorfönstret. Notera kortkommandot för Sök/Ersätt.
- ✓ g) Gör automatisk formattering av koden med hjälp av lämpligt editor-kortkommando. Notera skillnaderna. Vilket autoformatteringsprogram gör programmet lättare att läsa? Vilka manuella formatteringsprogram tycker du bör göras för att öka läsbarheten? Diskutera läsbarheten med en handledare.

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. Visa din lösning för en handledare.



1.10.2 Frivilliga extrauppgifter

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

a) Skriv in koden nedan i Kojos editor och kör upprepade gånger med den gröna play-knappen. Hur långt tid tar det för din dator att räkna till 4.4 miljarder?¹

```
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 < 1e8.toLong) { 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 rundor 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

¹Det går att göra ungefär en heltalsaddition per klockcykel per kärna. Den första elektroniska datorn Eniac hade en klockfrekvens motsvarande 5kHz. Björn Regnells dator har en i7-4790K som turboklockar på 4.4 MHz.

www.extremetech.com/computing/185512-overclocking-intels-core-i7-4790k-can-devils-canyon-fix-haswells-low-clock-speeds/2

din dator ”vilar”. Vad är det för poäng med att en processor kan variera sin klockfrekvens?)

```
1 > lscpu | grep MHz
```

c) Ändra i koden i uppgift a) så att **while**-loopen bara kör 5 gånger. Kör programmet med den *gula* play-kappen. 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:

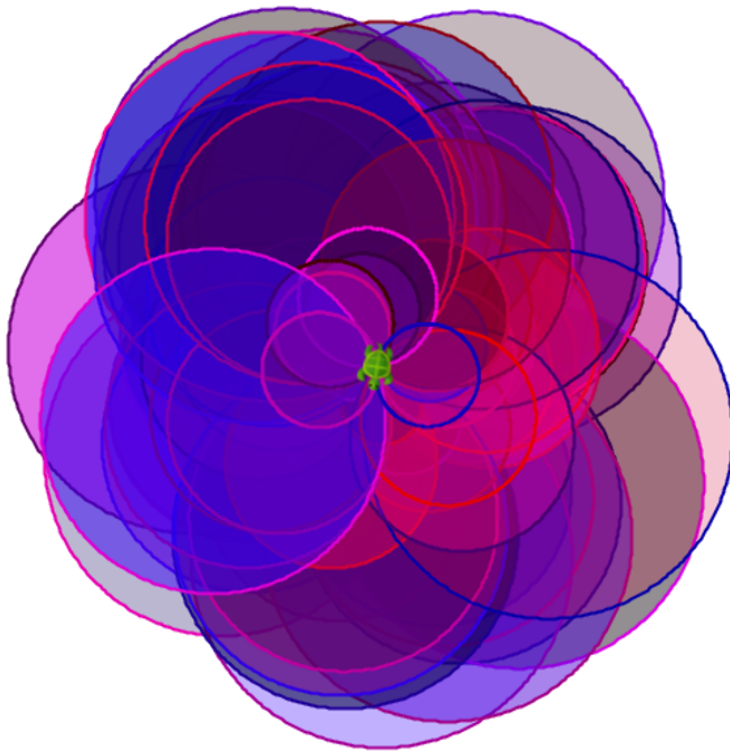
report(Int.MaxValue)

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

- ✓  Hur mycket snabbare går det att räkna med Int-variabler jämfört med Long-variabler? Visa svaret för en handledare.

Uppgift 7. Lek med färg i Kojo. Sök på internet efter dokumentationen för klassen `java.awt.Color` och studera vilka heltalsparametrar den sista konstruktorn i listan med konstruktorer tar för att skapa sRGB-färger. Om du högerklickar i editorn i Kojo och väljer "Välj färg..." får du fram färgväljaren.

```
1 scala> val c = new java.awt.Color(124,10,78,100)
2 c: java.awt.Color = java.awt.Color[r=124,g=10,b=78]
3
4 scala> c. // tryck på TAB
5 asInstanceOf   getColorComponents   getRGBComponents
6 brighter       getColorSpace      getRed
7 createContext  getComponents   getTransparency
8 darker        getGreen           instanceof
9 getAlpha       getRGB           toString
10 getBlue       getRGBColorComponents
11
12 scala> c.getAlpha
13 res3: Int = 100
```



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

- "Uppdrag med Kojo" som kan laddas ner här:
fileadmin.cs.lth.se/cs/Personal/Bjorn_Regnell/uppdrag.pdf
- "Programming Fundamentals with Kojo" som kan laddas ner här:
wiki.kogics.net/kojo-codeactive-books

Kapitel 2

Kodstrukturer

Koncept du ska lära dig denna vecka:

- ☐ Range
- ☐ Array
- ☐ Vector
- ☐ iterering
- ☐ for-uttryck
- ☐ map
- ☐ foreach
- ☐ algoritm vs implementation
- ☐ pseudokod
- ☐ algoritm: SWAP
- ☐ algoritm: SUM
- ☐ algoritm: MIN/MAX
- ☐ block
- ☐ namnsynlighet
- ☐ namnöverskuggning
- ☐ lokala variabler
- ☐ paket
- ☐ import
- ☐ filstruktur
- ☐ jar
- ☐ dokumentation
- ☐ programlayout
- ☐ JDK
- ☐ main i Java vs Scala
- ☐ `java.lang.System.out.println`

2.1 Övning: programs

Mål

- ☐ Kunna skapa samlingarna Range, Array och Vector med heltals- och strängvärden.
- ☐ Kunna indexera i en indexerbar samling, t.ex. Array och Vector.
- ☐ Kunna anropa operationerna size, mkString, sum, min, max på samlingar som innehåller heltal.
- ☐ Känna till grundläggande skillnader och likheter mellan samlingarna Range, Array och Vector.
- ☐ Förstå skillnaden mellan en for-sats och ett for-uttryck.
- ☐ Kunna skapa samlingar med heltalsvärden som resultat av enkla for-uttryck.
- ☐ Förstå skillnaden mellan en algoritm i pseudo-kod och dess implementation.
- ☐ Kunna implementera algoritmerna SUM, MIN/MAX på en indexerbar samling med en **while**-sats.
- ☐ Kunna köra igång enkel Scala-kod i REPL, som skript och som applikation.
- ☐ Kunna implementera och köra igång ett Java-program.
- ☐ Känna till några grundläggande syntaxskillnader mellan Scala och Java, speciellt variabeldeklarationer och indexering i Array.
- ☐ Förstå vad ett block är.
- ☐ Förstå vad en lokal variabel är.
- ☐ Förstå hur nästlade block påverkar namnsynlighet och namnöverskuggning.
- ☐ Förstå kopplingen mellan paketstruktur och klassfilstruktur.
- ☐ Kunna skapa en jar-fil.
- ☐ Kunna skapa dokumentation med scaladoc.

Förberedelser

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

2.1.1 Grunduppgifter

Uppgift 1. *Datastrukturen Range.* Evaluera nedan uttryck i Scala REPL. Vad har respektive uttryck för värde och typ?

- a) Range(1, 10)
- b) Range(1, 10).inclusive
- c) Range(0, 50, 5)
- d) Range(0, 50, 5).size

- e) `Range(0, 50, 5).inclusive`
- f) `Range(0, 50, 5).inclusive.size`
- g) `0.until(10)`
- h) `0 until (10)`
- i) `0 until 10`
- j) `0.to(10)`
- k) `0 to 10`
- l) `0.until(50).by(5)`
- m) `0 to 50 by 5`
- n) `(0 to 50 by 5).size`
- o) `(1 to 1000).sum`

Uppgift 2. *Datastrukturen Array.* Kör nedan kodrader i Scala REPL. Beskriv vad som händer.

- a) `val xs = Array("hej", "på", "dej", "!")`
- b) `xs(0)`
- c) `xs(3)`
- d) `xs(4)`
- e) `xs(1) + " " + xs(2)`
- f) `xs.mkString`
- g) `xs.mkString(" ")`
- h) `xs.mkString("(", ", ", ")")`
- i) `xs.mkString("Array(", ", ", ")")`
- j) `xs(0) = 42`
- k) `xs(0) = "42"; println(xs(0))`
- l) `val ys = Array(42, 7, 3, 8)`
- m) `ys.sum`
- n) `ys.min`
- o) `ys.max`
- p) `val zs = Array.fill(10)(42)`
- q) `zs.sum`

Uppgift 3. *Datastrukturen Vector.* Kör nedan kodrader i Scala REPL. Beskriv vad som händer.

- a) `val words = Vector("hej", "på", "dej", "!")`
- b) `words(0)`
- c) `words(3)`
- d) `words.mkString`
- e) `words.mkString(" ")`
- f) `words.mkString("(", ", ", ")")`

- g) `words.mkString("Ord(", ", ", ")")`
- h) `words(0) = "42"`
- i) `val numbers = Vector(42, 7, 3, 8)`
- j) `numbers.sum`
- k) `numbers.min`
- l) `numbers.max`
- m) `val moreNumbers = Vector.fill(10000)(42)`
- n) `moreNumbers.sum`
- o) Jämför med uppgift 2. Vad kan man göra med en Array som man inte kan göra med en Vector? 

Uppgift 4. *for*-uttryck. Evaluera nedan uttryck i Scala REPL. Vad har respektive uttryck för värde och typ?

- a) `for (i <- Range(1,10)) yield i`
- b) `for (i <- 1 until 10) yield i`
- c) `for (i <- 1 until 10) yield i + 1`
- d) `for (i <- Range(1,10).inclusive) yield i`
- e) `for (i <- 1 to 10) yield i`
- f) `for (i <- 1 to 10) yield i + 1`
- g) `(for (i <- 1 to 10) yield i + 1).sum`
- h) `for (x <- 0.0 to 2 * math.Pi by math.Pi/4) yield math.sin(x)`

Uppgift 5. Metoden *map* på en samling. Evaluera nedan uttryck i Scala REPL. Vad har respektive uttryck för värde och typ?

- a) `Range(0,10).map(i => i + 1)`
- b) `(0 until 10).map(i => i + 1)`
- c) `(1 to 10).map(i => i * 2)`
- d) `(1 to 10).map(_ * 2)`
- e) `Vector.fill(10000)(42).map(_ + 43)`

Uppgift 6. Metoden *foreach* på en samling. Kör nedan satser i Scala REPL. Vad händer?

- a) `Range(0,10).foreach(i => println(i))`
- b) `(0 until 10).foreach(i => println(i))`
- c) `(1 to 10).foreach{i => print("hej"); println(i * 2)}`
- d) `(1 to 10).foreach(println)`
- e) `Vector.fill(10000)(math.random).foreach(r => if (r > 0.99) print("pling!"))`

Uppgift 7. Algoritmen: SWAP.

- a) Skriv med *pseudo-kod* algoritmen SWAP. Beskriv på vanlig svenska, steg för steg, hur en variabel *temp* används för mellanlagring vid värdebytet:

Indata: två heltalsvariabler x och y
???

Utdata: variablerna x och y vars värden har bytt plats.

b) Implementerar algoritmen SWAP. Ersätt ??? nedan med satser separerade av semikolon:

```
1 scala> var (x, y) = (42, 43)
2 scala> ???
3 scala> println("x är " + x + ", y är " + y)
4 x är 43, y är 42
```

Uppgift 8. Skript. 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?
- c) Lägg till en sats sist i skriptet som skriver ut summan av de ett tusen stycken heltalen från och med 2 till och med 1001, så som visas nedan.

```
1 > scala hello-script.scala
2 hej skript
3 501500
```

- d) Ändra i `hello-script.scala` genom att införa **val** `n = args(0).toInt` och använd `n` som övre gräns för summeringen av de `n` första heltalen.

```
1 > scala hello-script.scala 5001
2 hej skript
3 12507501
```

- e) Vad blir det för felmeddelande om du glömmer ge programmet ett argument?

Uppgift 9. Applikation med main-metod. 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) Ändra i din kod så att kompilatorn ger följande felmeddelande:
Missing closing brace
- c) Varför behövs `main`-metoden?
- d) Vilket alternativ går snabbast att köra igång, ett skript eller en kompilerad applikation? Varför? Vilket alternativ kör snabbast när väl exekveringen är igång?



Uppgift 10. *Java-applikation.* Skapa med hjälp av en editor en fil med namn `Hi.java`.

```
> gedit Hi.java
```

Skriv dessa rader i filen:

```
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) Jämför signaturen för Java-programmets `main`-metod med signaturen för Scala-programmets `main`-metod. De betyder samma sak men syntaxen är olika. Beskriv skillnader och likheter i syntaxen.
- c) Vad blir det för felmeddelande om källkodsfilen och klassnamnet inte överensstämmer i ett Java-program?



Uppgift 11. *Algoritm: SUMBUG.* Nedan återfinns pseudo-koden för SUMBUG.

```

Indata : heltalet  $n$ 
Resultat : utskrift av summan av de första  $n$  heltalen
1  $sum \leftarrow 0$ 
2  $i \leftarrow 1$ 
3 while  $i \leq n$  do
4    $sum \leftarrow sum + 1$ 
5 end
6 skriv ut  $sum$ 

```

a) Kör algoritmen steg för steg med penna och papper, där du skriver upp hur värdena för respektive variabel ändras. Det finns en bugg i algoritmen. Vilken? Rätta buggen.

b) Skapa med hjälp av en editor filen `sumn.scala`. Implementera algoritmen SUM enligt den rättade pseudokoden och placera implementationen i en main-metod i ett objekt med namnet `sumn`. Du kan skapa indata n till algoritmen med denna deklaration i början av din main-metod:

```
val n = args(0).toInt
```

Vad ger applikationen för utskrift om du kör den med argumentet 8888?

```
> scalac sumn.scala
> scala sumn 8888
```

c) Kontrollera att din implementation räknar rätt genom att jämföra svaret med detta uttrycks värde, evaluerat i Scala REPL:

```
scala> (1 to 8888).sum
```

d) Implementera algoritmen SUM enligt pseudokoden ovan, men nu i Java. Skapa filen `SumN.java` och använd koden från uppgift 10 som mall för att deklarera den publika klassen `SumN` med en main-metod. Några tips om Java-syntax och standardfunktioner i Java:

- Alla satser i Java måste avslutas med semikolon.
- Heltalsvariabler deklareras med nyckelordet **int** (litet i).
- Typnamnet ska stå *före* namnet på variabeln. Exempel:
int sum = 0;
- Indexering i en array görs i Java med hakparenteser: `args[0]`
- I stället för Scala-uttrycket `args(0).toInt`, använd Java-uttrycket:
`Integer.parseInt(args[0])`
- **while**-satser i Scala och Java har samma syntax.
- Utskrift i Java görs med `System.out.println`

Uppgift 12. *Algorithm: MAXBUG.* Nedan återfinns pseudo-koden för MAXBUG.

```

Indata : Array args med strängar som alla innehåller heltal
Resultat: utskrift av största heltalet
1 max ← det minsta heltalet som kan uppkomma
2 n ← antalet heltal
3 i ← 0
4 while i < n do
5   | x ← args(i).toInt
6   | if (x > max) then
7   |   | max ← x
8   | end
9 end
10 skriv ut max

```

a) Kör med penna och papper. Det finns en bugg i algoritmen ovan. Vilken? Rätta buggen. 


b) Implementera algoritmen MAX (utan bugg) som en Scala-applikation. Tips:

- Det minsta Int-värdet som någonsin kan uppkomma: `Int.MinValue`
- Antalet element i *args* ges av: *args.size*

```

1 > gedit maxn.scala
2 > scalac maxn.scala
3 > scala maxn 7 42 1 -5 9
4 42

```

c) Skriv om algoritmen så att variabeln *max* initialiseras med det första talet i sekvensen. 

d) Implementera den nya algoritmvarianten från uppgift c och prova programmet. Vad händer om *args* är tom?

Uppgift 13. *Block, namnsynlighet, namnöverskuggning.* Kör nedan kod i Scala REPL eller i Kojo. Vad händer nedan? Varför?

- `val a = {1 + 1; 2 + 2; 3 + 3; 4 + 4}; println(a)`
- `val b = {1; 2; 3; {val b = 4; b + b; b + 1}}; println(b)`
- `{val a = 42; println(a)}`
- `{val a = 42}; println(a)`
- `{val a = 42; {val a = 43; println(a)}; println(a)}`
- `{var a = 42; {a = a + 1}; var a = 43}`
- `{var a = 42; {a = a + b; var b = 43}; println(a)}`
- `{var a = 42; {var b = 43; a = a + b}; println(a)}`
- `{var a = 42; {a = a + b; def b = 43}; println(a)}`
- `{object a{var b=42;object a{var a=43}};println(a.b+a.a.a)}`
-

```
{
  object a {
    var b = 42
    object a {
      var a = 43
    }
  }
  println(a.b + a.a.a)
}
```

l) Vad är fördelen med att namn deklarerade inne i ett block är lokala i stället för globala?


Uppgift 14. *Paket, **import** och klassfilstrukturer.* Med Java-8-plattformen kommer 4240 färdiga klasser, som är organiserade i 217 olika paket.¹

a) Vilka paket finns i paketet javax som börjar på s?

```
scala> javax.s //tryck på TAB-tangenten
```

b) Kör raderna nedan i REPL. Beskriv vad som händer för varje rad.

```
1 scala> import javax.swing.JOptionPane
2 scala> def msg(s: String) = JOptionPane.showMessageDialog(null, s)
3 scala> msg("Hej på dej!")
4 scala> def input(msg: String) = JOptionPane.showInputDialog(null, msg)
5 scala> input("Vad heter du?")
6 scala> import JOptionPane.{showOptionDialog => optDlg}
7 scala> def inputOption(msg: String, opt: Array[Object]) =
8     optDlg(null, msg, "Option", 0, 0, null, opt, opt(0))
9 scala> inputOption("Vad väljer du?", Array("Sten", "Sax", "Påse"))
```

 Vad hade du behövt ändra på efterföljande rader om import-satsen på rad 1 ovan ej hade gjorts?

d) Skapa med en editor filen paket.scala och kompilera. Rita en bild av hur katalogstrukturen ser ut.

```
package gurka.tomat.banan

package p1 {
  package p11 {
    object hello {
      def hello = println("Hej paket p1.p11!")
    }
  }
  package p12 {
    object hello {
      def hello = println("Hej paket p1.p12!")
    }
  }
}
```

¹Se Stackoverflow: [how-many-classes-are-there-in-java-standard-edition](#)

```

    }
  }
}

package p2 {
  package p21 {
    object hello {
      def hello = println("Hej paket p2.p21!")
    }
  }
}

object Main {
  def main(args: Array[String]): Unit = {
    import p1._
    p11.hello.hello
    p12.hello.hello
    import p2.{p21 => apelsin}
    apelsin.hello.hello
  }
}

```

```

1 > gedit paket.scala
2 > scalac paket.scala
3 > scala gurka.tomat.banan.Main
4 > ls -R

```

Uppgift 15. Skapa jar-filer och använda classpath

- Skriv kommandot `jar` i terminalen och undersök vad som finns för optioner. Se speciellt "Example 1." i hjälputskriften. Vilket kommando ska du använda för att packa ihop flera filer i en enda jar-fil?
- Som en fortsättning på uppgift 14, packa ihop biblioteket gurka i en jar-fil med nedan kommando, samt kör igång REPL med jar-filen på classpath.

```

1 > jar cvf mittpaket.jar gurka
2 > scala -cp mittpaket.jar
3 scala> gurka.tomat.banan.Main.main(Array())

```

Uppgift 16. Skapa dokumentation med scaladoc-kommandot

- Som en fortsättning på uppgift 14, kör nedan kommando i terminalen:

```

1 > scaladoc paket.scala
2 > ls
3 > firefox index.html # eller öppna index.html i valfri webbläsare

```

Vad händer?

b) Lägg till några fler metoder i något av objekten i filen `paket.scala` och lägg även till några dokumentationskommentarer. Kompilera om och kör. Generera om dokumentationen.

```
//... ändra i filen paket.scala

/** min paketedokumentationskommentar p2 */
package p2 {
  /** min paketedokumentationskommentar p21 */
  package p21 {
    /** ett hälsningsobjekt */
    object hello {
      /** en hälsningsmetod i p2.p21 */
      def hello = println("Hej paket p2.p21!")

      /** en metod som skriver ut tiden */
      def date = println(new java.util.Date)
    }
  }
}
```

```
1 > gedit paket.scala
2 > scalac paket.scala
3 > jar cvf mittpaket.jar gurka
4 > scala -cp mittpaket.jar
5 scala> gurka.tomat.banan.p2.p21.hello.date
6 scala> :q
7 > scaladoc paket.scala
8 > firefox index.html
```

2.1.2 Extrauppgifter: öva mer på grunderna

2.1.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 17. `ArrayBuffer` vs `Vector` vs `Array` och metoden `append`

Uppgift 18. Läs om krullparentheser och vanliga parenteser på stack overflow: [what-is-the-formal-difference-in-scala-between-braces-and-parentheses-and-when](#)

Uppgift 19. Bygg vidare på koden nedan och gör ett Sten-Sax-Påse-spel² som även meddelar vem som vinner. Koden fungerar att köra som den är, men funktionen `winnerMsg` är ej klar. *Tips:* Du kan använda modulo-räkning med `%`-operatoren för att avgöra vem som vinner.

²sv.wikipedia.org/wiki/Sten,_sax,_p%C3%A5se

```
object Rock {  
  import javax.swing.JOptionPane  
  import JOptionPane.{showOptionDialog => optDlg}  
  
  def inputOption(msg: String, opt: Vector[String]) =  
    optDlg(null, msg, "Option", 0, 0, null, opt.toArray[Object], opt(0))  
  
  def msg(s: String) = JOptionPane.showMessageDialog(null, s)  
  
  val opt = Vector("Sten", "Sax", "Påse")  
  
  def userChoice = inputOption("Vad väljer du?", opt)  
  
  def computerChoice = (math.random * 3).toInt  
  
  def winnerMsg(user: Int, computer: Int) = "??? vann!"  
  
  def main(args: Array[String]): Unit = {  
    var keepPlaying = true  
    while (keepPlaying) {  
      val u = userChoice  
      val c = computerChoice  
      msg("Du valde " + opt(u) + "\n" +  
        "Datorn valde " + opt(c) + "\n" +  
        winnerMsg(u, c))  
      if (u != c) keepPlaying = false  
    }  
  }  
}
```

Kapitel 3

Funktioner, Objekt

Koncept du ska lära dig denna vecka:

- | | |
|--|--|
| <input type="checkbox"/> definera funktion | <input type="checkbox"/> metod |
| <input type="checkbox"/> anropa funktion | <input type="checkbox"/> medlem |
| <input type="checkbox"/> parameter | <input type="checkbox"/> funktionsvärde |
| <input type="checkbox"/> returtyp | <input type="checkbox"/> funktionstyp |
| <input type="checkbox"/> värdeandrop | <input type="checkbox"/> äkta funktion |
| <input type="checkbox"/> namnanrop | <input type="checkbox"/> stegad funktion |
| <input type="checkbox"/> default-argument | <input type="checkbox"/> apply |
| <input type="checkbox"/> namngivna argument | <input type="checkbox"/> lazy val |
| <input type="checkbox"/> applicera funktion på alla element i en samling | <input type="checkbox"/> lokala funktioner |
| <input type="checkbox"/> procedur | <input type="checkbox"/> anonyma funktioner |
| <input type="checkbox"/> värdeanrop vs namnanrop | <input type="checkbox"/> lambda |
| <input type="checkbox"/> uppdelad parameterlista | <input type="checkbox"/> aktiveringspost |
| <input type="checkbox"/> skapa egen kontrollstruktur | <input type="checkbox"/> rekursion |
| <input type="checkbox"/> objekt | <input type="checkbox"/> basfall |
| <input type="checkbox"/> modul | <input type="checkbox"/> anropsstacken |
| <input type="checkbox"/> punktnotation | <input type="checkbox"/> objektheapen |
| <input type="checkbox"/> tillstånd | <input type="checkbox"/> cslib.window.SimpleWindow |

3.1 Övning: functions

Mål

- ☐ Kunna skapa och använda funktioner med en eller flera parametrar, default-argument, namngivna argument, och uppdelad parameterlista.
- ☐ Kunna använda funktioner som äkta värden.
- ☐ Kunna applicera en funktion på element i en samling.
- ☐ Förstå skillnader och likheter mellan en funktion och en procedur.
- ☐ Förstå skillnader och likheter mellan en värde-anrop och namnanrop.
- ☐ Kunna skapa en procedur i form av en enkel kontrollstruktur med fördröjd evaluering av ett block.
- ☐ Kunna skapa och använda objekt som moduler.
- ☐ Förstå skillnaden mellan äkta funktioner och funktioner med sidoeffekter.
- ☐ Kunna skapa och använda variabler med fördröjd initialisering och förstå när de är användbara.
- ☐ Kunna förklara hur nästlade funktionsanrop fungerar med hjälp av begreppet aktiveringspost.
- ☐ Kunna skapa och använda lokala funktioner, samt förstå nyttan med lokala funktioner.
- ☐ Känna till att funktioner är objekt med en apply-metod.
- ☐ Känna till stegade funktioner och kunna använda partiellt applicerade argument.
- ☐ Känna till rekursion och kunna förklara hur rekursiva funktioner fungerar med hjälp av anropsstacken.
- ☐ Känna till svansrekursion och att svansrekursiva funktioner kan optimeras till loopar.

Förberedelser

- ☐ Studera teorin i kapitel 3.



3.1.1 Grunduppgifter

Uppgift 1. *Definiera och anropa funktioner.* En funktion med två parametrar definieras med följande syntax i Scala:

```
def namn(parameter1: Typ1, parameter2: Typ2): Returtyp = returvärde
```

a) Definiera en funktion med namnet `öka` som har en heltalsparameter `x` och som returnerar `x + 1`. Ange returtypen explicit. Testa funktionen i REPL med argumentet 42.

```
1 scala> ??? // definiera funktionen öka
2 scala> öka(42)
3 43
```


- b) Vad har funktionen öka i föregående uppgift för returtyp?
- c) Vad gör kompilatorn om du utelämnar returtypen?
-  d) Varför kan det vara bra att ange returtypen explicit?
-  e) Vad är det för skillnad mellan parameter och argument?
- f) Vad har uttrycket öka(öka(öka(öka(42)))) för värde?
- g) Definera funktionen minska(x: Int): Int med returvärdet x - 1.
- h) Vad är värdet av uttrycket öka(minska(öka(öka(minska(minska(42)))))

Uppgift 2. *Funktion med flera parametrar.* Definiera i REPL två funktioner sum och diff med två heltalsparametrar som returnerar summan respektive differensen av argumenten:

```
def sum(x: Int, y: Int): Int = x + y
```

```
def diff(x: Int, y: Int): Int = x - y
```

Vad har nedan uttryck för värden? Förklara vad som händer.


- a) diff(0, 100)
- b) diff(100, add(42, 43))
- c) sum(sum(42, 43), diff(100, sum(0, 0)))
- d) sum(diff(Byte.MaxValue, Byte.MinValue), 1)

Uppgift 3. *Funktion med default-argument.* Förklara vad som händer här?

```
1 scala> def inc(i: Int, j: Int = 1) = i + j
2 scala> inc(42, 2)
3 scala> inc(42, 1)
4 scala> inc(42)
```

Uppgift 4. *Funktionsanrop med namngivna argument.*

```
1 scala> def skrivNamn(förnamn: String, efternamn: String) =
2     println("Namn: " + efternamn + ", " + förnamn)
3 scala> skrivNamn("Kim", "Robinson")
4 scala> skrivNamn(förnamn = "Viktor", efternamn = "Oval")
5 scala> skrivNamn(efternamn = "Triangelsson", förnamn = "Stina")
```

- a) Förklara vad som händer ovan?
-  b) Vad är fördelen med namngivna argument?

Uppgift 5. *Applicera en funktion på elementen i en samling.* Använd dina funktioner öka och minska från uppgift 1. Vad har nedan uttryck för värde? Förklara vad som händer.

- a) `for (i <- 0 to 4) yield öka(i)`
- b) `for (i <- 1 to 5) yield minska(i)`
- c) `(0 to 4).map(i => öka(i))`
- d) `(1 to 5).map(i => minska(i))`
- e) `(0 to 4).map(öka)`

- f) `(1 to 5).map(minska)`
- g) `Vector(12, 3, 41, -8).map(öka)`
- h) `Vector(12, 3, 41, -8).map(öka).map(minska).map(minska)`

Uppgift 6. En funktion som inte returnerar något intressant värde, men som anropas för det den *gör* kallas **procedur**. Definiera följande procedur i REPL:

```
def tUvirks(msg: String) = println(msg.reverse)
```

Vad skriver nedan satser ut? Förklara vad som händer.

- a) `println("sallad".reverse)`
- b) `tUvirks("sallad")`
- c) `val x = tUvirks("sallad"); println(x)`
- d) `def enhetsvärdet = (); println(enhetsvärdet)`
- e) `def bortkastad: Unit = 1 + 1; println(bortkastad)`
- f) `def bortkastad2 = {val x = 1 + 1}; println(bortkastad2)`
- g) Varför är det bra att explicit ange Unit som returtyp för procedurer?



Uppgift 7. Värdeanrop och namnanrop (fördröjd evaluering, "lata" argument).

Deklarera nedan funktioner i REPL eller Kojo.

```
def snark: Int = {print("snark "); Thread.sleep(1000); 42}
def callByValue(x: Int) = x + x
def callByName(x: => Int) = x + x
```

Evaluera nedan uttryck. Förklara vad som händer.

- a) `snark`
- b) `snark; snark; snark`
- c) `callByValue(1)`
- d) `callByName(1)`
- e) `callByValue(snark)`
- f) `callByName(snark)`
- g) Förklara vad som händer här:

```
1 scala> def görDetta(block: => Unit) = block
2 scala> görDetta(println("hej"))
3 scala> görDetta{println("goddag")}
4 scala> görDetta{println("hej"); println("svejs")}
5 scala> def görDettaTvåGånger(block: => Unit) = {block; block}
6 scala> görDettaTvåGånger{println("goddag")}
```

Uppgift 8. Uppdelad parameterlista. Man kan dela upp parametrarna till en funktion i flera parameterlistor. Förklara vad som händer här:

```
1 scala> def add(a: Int)(b: Int) = a + b
2 scala> add(22)(20)
3 scala> add(22)(add(1)(19))
```

Uppgift 9. Skapa din egen kontrollstruktur. Använd fördröjd evaluering och stegad funktion och skapa din egen loop-konstruktion.

```
1 scala> def upprepa(n: Int)(block: => Unit) = {
2     var i = 0
3     while (i < n) {block; i = i + 1}
4 }
5 scala> upprepa(10)(println("hej"))
6 scala> upprepa(1000){
7     val tärning = (math.random * 6 + 1).toInt
8     print(tärning + " ")
9 }
```

Förklara vad som händer ovan. (Det är så som upprepa i Kojo är definierad.)

Uppgift 10. Funktion som värde. Funktioner är äkta värden i Scala.

a) Förklara vad som händer nedan. Notera understrecket på rad 4:

```
1 scala> def inc(x: Int): Int = x + 1
2 scala> inc(42)
3 scala> Vector(12, 3, 41, -8).map(inc)
4 scala> val f = inc _
5 scala> Vector(12, 3, 41, -8).map(f)
```

b) Vad händer om du bara skriver **val** f = inc utan understreck?

c) På liknande sätt som i uppgift a: definiera en funktion dec som i stället minskar med 1. Deklarera ett funktionsvärde g som tilldelas funktionen dec och kör sedan g på varje element i Vector(12, 3, 41, -8) med metoden map.



d) Vad har variablerna f och g ovan för typ?

e) Förklara vad som händer nedan. Vad får d och h för värde?

```
1 scala> def räkna(x: Int, f: Int => Int) = f(x,y)
2 scala> def dubbla(x: Int) = 2 * x
3 scala> def halva(x: Int) = x / 2
4 scala> val d = räkna(42, dubbla)
5 scala> val h = räkna(42, halva)
```

Uppgift 11. Stegade funktioner ("Curry-funktioner"). Förklara vad som händer nedan.

```
1 scala> def sum(a: Int)(b: Int) = a + b
2 scala> sum(1)(2)
3 scala> val f = sum(42) _
4 scala> f(1)
5 scala> val inc = sum(1) _
6 scala> val dec = sum(-1) _
7 scala> inc(42)
8 scala> dec(42)
```

Uppgift 12. Objekt som moduler.

a) Lär dig följande terminologi utantill:

- Ett objekt som samlar funktioner och variabler kallas även en **modul**.
- Funktioner i objekt kallas även **metoder**.
- Variabler och metoder i objekt kallas **medlemmar**.
- Moduler kan i sin tur innehålla moduler, i godtyckligt nästlingsdjup.
- Man kommer åt innehållet i en modul med **punktnotation**.
- Med **import** slipper man punktnotation.
- Ett objekt med variabler sägs ha ett **tillstånd**.

b) Deklarera modulerna stringstat och Test nedan i REPL eller i Kojo.

```
object stringstat {
  object stringfun {
    def sentences(s: String): Array[String] = s.split('.')
    def words(s: String): Array[String] = s.split(' ')
    def countWords(s: String): Int = words(s).size
    def countSentences(s: String): Int = sentences(s).size
  }


  object statistics {
    var history = ""
    def printFreq(s: String): Unit = {
      println("\n---- Frekvenser ----")
      println("Antal tecken:   " + s.size)
      println("Antal ord:         " + stringfun.countWords(s))
      println("Antal meningar:  " + stringfun.countSentences(s))
      history = history + " " + s
    }
    def printTotal: Unit = printFreq(history)
  }
}

object Test {
  import stringstat._
  def apply(n: Int = 42): Unit = {
    val s1 = "Fem myror är fler än fyra elefanter. Ät gurka."
    val s2 = "Galaxer i mina braxer. Tomat är gott. Hejsan."
    statistics.printFreq(s1 * n)
    statistics.printFreq(s2 * n)
    statistics.printTotal
  }
}
```

- c) Anropa Test() och förklara vad som händer. Vad skrivs ut?
- d) Vilket av objekten i modulen stringstat har tillstånd och vilket av objekten är tillståndslöst? Vad består tillståndet av?

Uppgift 13. Äkta funktioner. En **äkta funktion** ger alltid samma resultat med samma argument.

```
object inSearchOfPurity {
  var x = 0
  val y = x
  def inc(i: Int) = i + 1
  def oink(i: Int) = {x = x + i; "Pig says oink " + x}
  def addX(i: Int): Int = x + i
  def addY(i: Int): Int = y + i
  def isPalindrome(s: String): Boolean = s == s.reverse
  def rnd(min: Int, max: Int) = math.random * max + min
}
```

-  a) Vilka funktioner i objektet `inSearchOfPurity` är äkta funktioner?
- b) Anropa de funktioner som inte är äkta i REPL och demonstrera med exempel att de kan ge olika resultat för samma argument.
- c) Vad objektets är tillstånd efter dina körningar i uppgift b?
- d) Vilken del av tillståndet i objektet är oföränderligt?

Uppgift 14. Funktioner är objekt med en `apply`-metod.

- a) Förklara vad som händer här:

```
1 scala> object plus { def apply(x: Int, y: Int) = x + y }
2 scala> plus.apply(42,43)
3 scala> plus(42, 43)
4 scala> val add: (Int, Int) => Int = (x, y) => x + y
5 scala> add(42, 42)
6 scala> add.    // tryck på TAB
7 scala> add.apply(42, 42)
8 scala> val inc = add.curried(1)
9 scala> inc(42)
```

- b) Definiera i REPL ett objekt som heter `Slumptal` som har en `apply`-metod som tar två heltalsparametrar `a` och `b` och som med hjälp av `math.random` returnerar ett slumpal i intervallet $[a, b]$. Anropa objektets `apply`-metod med `(1 to 100).foreach(println(???))`, där `???` ersätts först med punktnotation och sedan med funktionsapplieringssyntax.

Uppgift 15. *Fördröjd initialisering ("lata" variabler).*

- a) Förklara vad som händer här:

```
1 scala> val olat = 42
2 scala> lazy val lat = 42
3 scala> println(lat)
4 scala> val nu = {Thread.sleep(1000); println("nu"); 42}
5 scala> lazy val sen = {Thread.sleep(1000); println("sen"); 42}
6 scala> def igen = {Thread.sleep(1000); println("hver gang"); 42}
7 scala> println(nu)
8 scala> println(sen)
9 scala> println(igen)
10 scala> println(nu)
```

```

11 scala> println(sen)
12 scala> println(igen)
13 scala> object m {lazy val stor = Array.fill(1e9.toInt)(liten); val liten = 42}
14 scala> m.liten
15 scala> m.stor

```

b) Vad är skillnaden mellan **val**, **lazy val** och **def**, vad gäller *när* evalueringen sker? 

c) Förklara vad som händer här:

```

1 scala> object objektÄrLata { val sen = { println("nu!"); 42 } }
2 scala> objektÄrLata
3 scala> objektÄrLata.sen
4 scala> {val x = y; val y = 42}
5 scala> object buggig {val a = b; val b = 42}
6 scala> buggig.a
7 scala> object funkar {lazy val a = b; val b = 42}
8 scala> funkar.a
9 scala> object nowarning {val many = Array.fill(10)(one); val one = 1}
10 scala> nowarning.many

```

d) Med ledning av uppgift a och uppgift c, beskriv två olika situationer när kan man ha nytta av **lazy val**? 

Uppgift 16. Aktiveringspost. Antag att vi bara kan addera eller subtrahera med ett. Då kan man ändå skapa en additionsfunktion på nedan (ganska omständliga) sätt. Skriv nedan program i en editor, kompilera och exekvera.

```


object Count {
  def inc(x: Int) = {println("inc[x = " + x + "]); x + 1}
  def dec(x: Int) = {println("dec[x = " + x + "]); x - 1}

  def add(x: Int, y: Int) = {
    println("add[x = " + x + ", y = " + y + "])
    var result = x;
    var i = 0;
    while (i < math.abs(y)){
      result = if (y >= 0) inc(result) else dec(result)
      i = i + 1
    }
    result
  }

  def main(args: Array[String]): Unit = {
    val x = inc(dec(inc(0)))
    println(x)
    val y = add(1, add(1, add(1, -2)))
    println(y)
  }
}

```

a) Vad skrivs ut? Förklara vad som händer.

 Rita hur anropsstacken förändras under exekveringen av main-metoden.


Uppgift 17. *Lokala funktioner.* Skapa nedan program i en editor, kompilera och exekvera. I programmet nedan har metoden `add` två lokala funktioner som skiljer sig från metoderna med samma namn.

```
object Count {
  def inc(x: Int) = x + 1
  def dec(x: Int) = x - 1

  def add(x: Int, y: Int) = {
    def inc(x: Int) = {println("inc[x = " + x + "]); x + 1}
    def dec(x: Int) = {println("dec[x = " + x + "]); x - 1}
    println("add[x = " + x + ", y = " + y + "]")
    var result = x;
    var i = 0;
    while (i < math.abs(y)){
      result = if (y >= 0) inc(result) else dec(result)
      i = i + 1
    }
    result
  }

  def main(args: Array[String]): Unit = {
    val x = inc(dec(inc(0)))
    println(x)
    val y = add(1, add(1, add(1, -2)))
    println(y)
  }
}
```

a) Vad skrivs ut? Förklara vad som händer.

 Vilka fördelar finns med lokala funktioner?

Uppgift 18. *Anonyma funktioner.* Vi har flera gånger sett syntaxen `i => i + 1`, till exempel i en loop `(1 to 10).map(i => i + 1)` där funktionen `i => i + 1` appliceras på alla heltal från 1 till och med 10. Funktionen `i => i + 1` kallas en **anonym** funktion, eftersom den inte har något namn, till skillnad från `def öka(i: Int): Int = i + 1`, som har namnet `öka`.

Anonyma funktioner kallas även för *funktionslitteraler* eller *lambda*.

Det finns ett ännu kortare sätt att skriva en anonym funktion om den bara använder sin parameter en enda gång, med understreck `_ + 1` som expanderas av kompilatorn till `ngtnamn => ngtnamn + 1` (namnet på parametern spelar ingen roll; kompilatorn väljer något eget, internt namn).

a) Förklara vad som händer nedan. Vad blir resultatet av varje uttryck?

```
1 scala> (1 to 4).map(i => i + 1)
2 scala> (1 to 4).map(_ + 1)
3 scala> (1 to 4).map(math.pow(2, _))
4 scala> (1 to 4).map(math.pow(_, 2))
5 scala> (1 to 4).map(i => i.toString)
6 scala> (1 to 4).map(_.toString)
```

b) Vilken typ kommer kompilatorn att härleda för de anonyma funktionerna i argumenten till metoden map på rad 1–6 i uppgiften ovan? Vad använder kompilatorn för information i dessa exempel för att härleda funktionstyperna?



c) Vilka felmeddelande ger kompilatorn när den inte kan lista ut att funktionsliterals nedan har typen Int => Int?



```
1 scala> val inc = i => i + 1
2 scala> val inc = (i: Int) => i + 1
3 scala> (1 to 10).map(inc)
4 scala> val dec = _ - 1
5 scala> val dec: Int => Int = _ - 1
6 scala> (1 to 10).map(dec)
```

Uppgift 19. Rekursion. En rekursiv funktion anropar sig själv.

a) Förklara vad som händer nedan.

```
1 scala> def countdown(x: Int): Unit = if (x > 0) {println(x); countdown(x - 1)}
2 scala> countdown(10)
3 scala> countdown(-1)
4 scala> def finalCountdown(x: Byte): Unit =
5     {println(x); Thread.sleep(100); finalCountdown((x-1).toByte); 1 / x}
6 scala> finalCountdown(Byte.MaxValue)
```

b) Vad händer om du gör satsen som riskerar division med noll *före* det rekursiva anropet i funktionen finalCountdown ovan?

c) Förklara vad som händer nedan. Varför tar sista raden längre tid än näst sista raden?

```
1 scala> def signum(a: Int): Int = if (a >= 0) 1 else -1
2 scala> def add(x: Int, y: Int): Int =
3     if (y == 0) x else add(x + 1, y - signum(y))
4 scala> add(100, 100)
5 scala> add(Int.MaxValue, 0)
6 scala> add(0, Int.MaxValue)
```

3.1.2 Extrauppgifter: öva mer på grunderna

Uppgift 20. Visa anropsstacken genom att kasta undantag.

3.1.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 21. *Kolla bajtkoden.*

```
1 scala> def plusxy(x: Int, y: Int) = x + y
2 scala> :javap plusxy
```

a) Leta upp raden `public int plusxy(int, int);` och studera koden efter Code: och försök gissa vilken instruktion som utför själva additionen.

b) Lägg till en parameter till:

```
def plusxyz(x: Int, y: Int, z: Int) = x + y + z
```

och studera bajtkoden med `:javap plusxyz`. Vad skiljer bajtkoden mellan `plusxy` och `plusxyz`?

 Läs om bajtkod här: en.wikipedia.org/wiki/Java_bytecode. Vad betyder den inledande bokstaven i additionsinstruktionen?

Uppgift 22. *Undersök svansrekursion genom att kasta undantag.* Förklara vad som händer. Kan du hitta bevis för att kompilatorn kan optimera rekursionen till en vanlig loop?

```
1 scala> def explode = throw new Exception("BANG!!!")
2 scala> explode
3 scala> lastException.printStackTrace
4 scala> def countdown(n: Int): Unit =
5     if (n == 0) explode else countdown(n-1)
6 scala> countdown(10)
7 scala> lastException.printStackTrace
8 scala> def countdown2(n: Int): Unit =
9     if (n == 0) explode else {countdown2(n-1); print("no tailrec")}
10 scala> countdown2(10)
11 scala> countdown2(1000)
12 scala> lastException
13 scala> lastException.getStackTrace.size
14 scala> :javap countdown
15 scala> :javap countdown2
```

Uppgift 23. *@tailrec-annotering.* Du kan be kompilatorn att ge felmeddelande om den inte kan optimera koden till en loop och därmed öka prestanda och undvika en överfull anropsstack (eng. *stack overflow*). Prova nedan rader i REPL och förklara vad som händer.

```
1 scala> def countNoTailrec(n: Long): Unit =
2     if (n <= 0L) println("Klar! " + n) else {countNoTailrec(n-1L); ()}
3 scala> countNoTailrec(1000L)
4 scala> countNoTailrec(100000L)
5 scala> import scala.annotation.tailrec
6 scala> @tailrec def countNoTailrec(n: Long): Unit =
7     if (n <= 0L) println("Klar! " + n) else {countNoTailrec(n-1L); ()}
8 scala> @tailrec def countTailrec(n: Long): Unit =
9     if (n <= 0L) println("Klar! " + n) else countTailrec(n-1L)
10 scala> countTailrec(1000L)
```

```
11 scala> countTailrec(100000L)
12 scala> countTailrec(Int.MaxValue.toLong * 2L)
```

3.2 Laboration: simplewindow

Mål

- ☐ Att lära sig.

Förberedelser

- ☐ Att göra.

3.2.1 Obligatoriska uppgifter

Uppgift 1. En labbuppgiftsbeskrivning.

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

3.2.2 Frivilliga extrauppgifter

Uppgift 2. En labbuppgiftsbeskrivning.

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

Kapitel 4

Datastrukturer

Koncept du ska lära dig denna vecka:

- | | |
|--|---|
| <input type="checkbox"/> attribut (fält) | <input type="checkbox"/> List |
| <input type="checkbox"/> medlem | <input type="checkbox"/> Vector |
| <input type="checkbox"/> metod | <input type="checkbox"/> Set |
| <input type="checkbox"/> tupel | <input type="checkbox"/> Map |
| <input type="checkbox"/> klass | <input type="checkbox"/> typparameter |
| <input type="checkbox"/> Any | <input type="checkbox"/> generisk samling som parameter |
| <input type="checkbox"/> instanceof | <input type="checkbox"/> översikt samlingsmetoder |
| <input type="checkbox"/> toString | <input type="checkbox"/> översikt strängmetoder |
| <input type="checkbox"/> case-klass | <input type="checkbox"/> läsa/skriva textfiler |
| <input type="checkbox"/> Complex | <input type="checkbox"/> Source.fromFile |
| <input type="checkbox"/> Rational | <input type="checkbox"/> java.nio.file |
| <input type="checkbox"/> föränderlighet vs oföränderlighet | |

4.0.3 Att göra denna vecka

4.1 Denna vecka: Fatta datastrukturer

- Läs teori
- Gör övning data
- Gör lab ???

4.1.1 Olika sorters datastrukturer

4.2 Olika sätt att skapa datastrukturer

- Tupler
 - samla n st datavärden i element **-1**, **-2**, ... $-n$
 - elementen kan vara av **olika** typ
- Klasser
 - samlar data i **attribut** med (väl valda!) namn
 - attributen kan vara av **olika** typ
 - definierar även metoder som använder attributen (operationer på data)
- Samlingar
 - speciella klasser som samlar data i element av **samma** typ
 - finns ofta *många* färdiga **bra-att-ha-metoder**

4.2.1 Tupler

4.3 Vad är en tupel?

`("hej", 42, math.Pi)` är en 3-tuple av denna typ:
`(String, Int, Double)`

4.4 Övning: data

Mål

- ☐ Kunna skapa och använda tupler, som variabelvärden, parametrar och returvärden.
- ☐ Förstå skillnaden mellan ett objekt och en klass och kunna förklara betydelsen av begreppet instans.
- ☐ Kunna skapa och använda attribut som medlemmar i objekt och klasser och som klassparametrar.
- ☐ Beskriva innebörden av och syftet med att ett attribut är privat.
- ☐ Kunna byta ut implementationen av metoden `toString`.
- ☐ Kunna skapa och använda en objektfabrik med metoden `apply`.
- ☐ Kunna skapa och använda en enkel case-klass.
- ☐ Kunna använda operatornotation och förklara relationen till punktnotation.
- ☐ Förstå konsekvensen av uppdatering av föränderlig data i samband med multipla referenser.
- ☐ Känna till och kunna använda några grundläggande metoder på samlingar.
- ☐ Känna till den principiella skillnaden mellan `List` och `Vector`.
- ☐ Kunna skapa och använda en oföränderlig mängd med klassen `Set`.
- ☐ Förstå skillnaden mellan en mängd och en sekvens.
- ☐ Kunna skapa och använda en nyckel-värde-tabell, `Map`.
- ☐ Förstå likheter och skillnader mellan en `Map` och en `Vektor`.

Förberedelser

- ☐ Studera teorin i kapitel 4.

4.4.1 Grunduppgifter

Uppgift 1. *En enkel datastruktur: tupel.* Du kan samla olika data i en tupel. Du kommer åt värdena med en metod som har namnet understreckt följt av ordningsnumret.

```
1 scala> val namn = ("Pippi", "Långstrump")
2 scala> namn._1
3 scala> namn._2
4 scala> println("Förnamn: " + namn._1 + "\nEfternamn: " + namn._2)
```

- a) Definiera en oföränderlig variabel med namnet `pt` som representerar en punkt med x-koordinaten 15.9 och y-koordinaten 28.9. Använd sedan `math.hypot` för att ta reda på avståndet från origo till punkten. Vad blir svaret?
- b) Du kan dela upp en tupel i sina beståndsdelar så här:

```
scala> val (förnamn, efternamn) = ("Ronja", "Rövardotter")
```

Dela upp din punkt `pt` i sina beståndsdelar och kalla delarna `x` och `y`

c) Värdena i en tupel kan ha olika typ.

```
scala> val creature = ("Doktor", "Krokodil", 65.0, false)
scala> val (title, name, weight, isHuman) = creature
```

Vilken typ har 4-tupeln `creature` ovan?

d) Tupler kan ingå i samlingar.

```
scala> val pts = Vector((0.0, 0.0), (1.0, 0.0), (1.0, 1.0), (0.0, 1.0))
scala> pts.foreach(println)
```

Vilken typ har vektorn `pts` ovan?

e) För 2-tupler finns ett kortare skrivsätt:

```
scala> ("Skåne", "Malmö")
scala> "Skåne" -> "Malmö"
scala> val huvudstäder = Vector("Sverige" -> "Stockholm", "Norge" -> "Oslo")
```

Lägg till fler huvudstäder i vektorn ovan.

f) Funktioner kan ta tupler som parametrar.

```
1 scala> def length(pt: (Double, Double)) = math.hypot(pt._1, pt._2)
2 scala> length((3.0, 4.0))
3 scala> length(3.0, 4.0) //kompilatorn lägger till parenteserna innan anrop
```

Applicera funktionen `length` ovan på alla tupler i samlingen `pts` från uppgift d med `map`. Vad får resultatet för värde och typ?

g) Funktioner kan ge tupler som resultat.

```
1 scala> def div(a: Int, b: Int) = (a / b, a % b)
2 scala> div(10, 3)
3 scala> (div(9,2), div(10,2))
4 scala> (div(9,2)._2, div(10,2)._2)
5 scala> val nOdd = (1 to 10).map(i => div(i, 2)._2).sum
```

Förklara vad som händer ovan. Använd `div` ovan för att ta reda på hur många udda tal finns det i intervallet `[1234,3456]`.

h) En tupel med n värden kallas n -tupel. Om man betraktar enhetsvärdet `()` som en tupel, vad kan man då kalla detta värde?

Uppgift 2. *Objekt med attribut (fält).* Ett objekt kan samla data som hör ihop och på så sätt skapa en datastruktur. Data i ett objekt kallas *attribut* eller *fält*, (eng. *field*). Objekt som samlar enbart data kallas även *post* (eng. *record*).

```
scala> object mittKonto { var saldo = 0; val nummer = 12345L }
```

a) Skriv en sats som sätter in ett slumpmässigt belopp mellan 0 och en miljon på `mittKonto` ovan med hjälp av punktnotation och tilldelning.


b) Vad händer om du försöker ändra attributet `nummer`?


Uppgift 3. *Klass med attribut.* Om du vill ha många objekt av samma typ, kan du använda en **klass**. På så sätt kan man skapa många datastrukturer av samma typ men med olika innehåll. Man skapar nya objekt med nyckelordet **new** följt av klassens namn. Klassen utgör en "mall" för objektet som skapas. Ett objekt som skapas med **new** Klassnamn kallas även en **instans** av klassen Klassnamn. Nedan skapas en datastruktur Konto som samlar data om ett bankkonto. Poster av typen Konto håller reda på hur mycket pengar det finns på kontot och vilket kontonumret är:

```

1 scala> class Konto {
2     var saldo = 0
3     var nummer = 0L
4 }
5 scala> val k1 = new Konto
6 scala> val k2 = new Konto
7 scala> k1.saldo = 1000
8 scala> k1.nummer = 12345L
9 scala> k2.saldo = 2000
10 scala> k2.nummer = 67890L
11 scala> println("Konto: " + k1.nummer + " Saldo:" k1.saldo)
12 scala> println("Konto: " + k2.nummer + " Saldo:" k2.saldo)

```

a) Rita hur minnessituationen ser ut efter att ovan rader har exekverats. 

 b) Vad hade det fått för konsekvenser om attributet nummer vore oföränderligt i klassen ovan? (Jämför med objektet mittKonto.)

Uppgift 4. *Klass med attribut som parametrar.* Om man vill ge attributen initialvärden när objektet skaps med **new** kan placera attributen i en parameterlista till klassen. Kod som körs när objektet skapas och attributen tilldelas sina initialvärden, kallas **konstruktör** (eng. *constructor*).

```

1 scala> class Konto(var saldo: Int, val nummer: Long)
2 scala> val k = new Konto(0, 12345L)
3 scala> println("Konto: " + k.nummer + " Saldo:" k.saldo)
4 scala> println(k)
5 scala> k.toString


```

a) Den två sista raderna ovan skriver ut den identifierare som JVM använder för att hålla reda på objektet i sina interna datastrukturer. Vad skrivs ut?

b) Skapa ännu en instans av klassen Konto med samma saldo och nummer som k ovan och spara den i **val** k2 och undersök dess objektidentifierare. Får objekten k och k2 olika objektidentifierare?

c) Sätt in olika belopp på respektive konto.

d) Vad händer om du försöker ändra attributet nummer?

 e) Ibland räcker det fint med en tupel, men ofta vill man ha en klass istället. Beskriv några fördelar med en Konto-klassen ovan jämfört med en tupel av typen (Int, Long).

```

scala> var k3 = (0, 12345L)
scala> k3 = (k3._1 + 100, k3._2)

```

Uppgift 5. Publikt versus privat attribut. Man kan förhindra att ett attribut syns utanför klassen med hjälp av nyckelordet **private**.

```
1 scala> class Konto1(val nummer: Long){ var saldo = 0 }
2 scala> val k1 = new Konto1(12345678901L)
3 scala> k1.nummer
4 scala> k1.saldo += 1000
5 scala> class Konto2(val nummer: Long){ private var saldo = 0 }
6 scala> val k2 = new Konto2(12345678901L)
7 scala> k2.nummer
8 scala> k2.saldo += 1000
```

- Vad händer ovan?
- Gör en ny version av klassen Konto enligt nedan:

```
class Konto(val nummer: Long){
  private var saldo = 0
  def in(belopp: Int): Unit = {saldo += belopp}
  def ut(belopp: Int): Unit = {saldo -= belopp}
  def show: Unit =
    println("Konto Nr: " + nummer + " saldo: " + saldo)
}

object Main {
  def main(args: Array[String]): Unit = {
    val k = new Konto(1234L)
    k.show
    k.in(1000)
    println("Uttag: " + k.ut(500))
    println("Uttag: " + k.ut(1000))
    k.show
  }
}
```

- Spara koden i en fil, kompilera och kör. Testa även vad som händer om du försöker komma åt attributet saldo i main-metoden med t.ex. `println(k.saldo)` eller `k.saldo += 1000`.
- Vi ska nu förhindra överuttag. Ändra i metoden `ut` så att den får signaturen `ut(belopp: Int): (Int Int) = ???` och implementera `ut` så att den returnerar både beloppet man verkligen kan ta ut och kvarvarande saldo. Om man försöker ta ut mer än det finns på kontot så ska saldot bli 0 och man får bara ut det som finns kvar. Spara, kompilera, kör.
- Förbättra metoderna `in` och `ut` så att man inte kan sätta in eller ta ut negativa belopp.
- Vad är fördelen med att göra föränderliga attribut privata och bara påverka deras värden indirekt via metoder?

Uppgift 6. Vilken typ har ett objekt? Objektets typ bestäms av klassen. Vid tilldelning måste typerna passa ihop.

a) Vilka rader nedan ger felmeddelande? Hur lyder felmeddelandet?

```
1 scala> class Punkt(val x: Double, val y: Double)
2 scala> val pt: Punkt = new Punkt(10.0, 10.0)
3 scala> val i: Int = pt.x
4 scala> val (x: Double, y: Double) = (pt.x, pt.y)
5 scala> val p: Double = new Punkt(5.0, 5.0)
6 scala> val p = new Punkt(5.0, 5.0): Double
7 scala> val p = new Punkt(5.0, 5.0): Punkt
8 scala> pt: Punkt
```

b) Man kan undersöka om ett objekt är av en viss typ med metoden `isInstanceOf[Typnamn]`. Vad ger nedan anrop av metoden `isInstanceOf` för värde?

```
1 scala> class Punkt(val x: Double, val y: Double)
2 scala> val pt: Punkt = new Punkt(1.0, 2.0)
3 scala> pt.isInstanceOf[Punkt]
4 scala> pt.isInstanceOf[Double]
5 scala> pt.x.isInstanceOf[Punkt]
6 scala> pt.x.isInstanceOf[Double]
7 scala> pt.x.isInstanceOf[Int]
```

Uppgift 7. Any. Alla klasser är också av typen Any. Alla klasser får därmed med sig några gemensamma metoder som finns i den fördefinierade klassen Any, däribland metoderna `isInstanceOf` och `toString`. Vad blir resultatet av respektive rad nedan? Vilken rad ger ett felmeddelande?

```
1 scala> class Punkt(val x: Double, val y: Double)
2 scala> val pt: Punkt = new Punkt(1.0, 2.0)
3 scala> pt.isInstanceOf[Punkt]
4 scala> pt.isInstanceOf[Any]
5 scala> pt.x.toString
6 scala> println(pt.x)
7 scala> val a: Any = pt
8 scala> println(a.x)
9 scala> a.toString
10 scala> p1.y.toString
11 scala> a.y.toString
```

Uppgift 8. Byta ut metoden `toString`. I klassen Any finns metoden `toString` som skapar en strängrepresentation av objektet. Du kan byta ut metoden `toString` i klassen Any mot din egen implementation. Man använder nyckelordet **override** när man vill byta ut en metodimplementation.

```
1 scala> class Punkt(val x: Double, val y: Double) {
2     override def toString: String = "[x=" + x + ",y=" + y + "]"
3 }
4 scala> val pt = new Punkt(1.0, 42.0)
5 scala> pt.toString
```

```
6 scala> println(pt)
```

- Vad händer egentligen på sista raden ovan?
- Omdefiniera `toString` så att den ger en sträng på formen `Punkt(1.0, 42.0)`.
- Vad händer om du utelämnar nyckelordet **override** vid omdefiniering?

Uppgift 9. *Objektfabrik med apply-metod.* Man kan ordna så att man slipper skriva **new** med ett s.k. *fabriksobjekt* (eng. *factory object*).

```
class Pt(val x: Double, y: Double) {
  override def toString: String = "Pt(x=" + x + ",y=" + y + ")"
}
object Pt {
  def apply(x: Double, y: Double): Pt = new Pt(x, y)
}
```

- Skriv satser som använder metoden `apply` i fabriksobjektet **object** `Pt` för att skapa flera olika punkter.
- Ge `apply`-metoden default-argument 0.0 för både `x` och `y` så att `Pt()` skapar en punkt i origo.
- Skapa en klass `Rational` som representerar rationellt tal som en kvot mellan två heltal. Ge klassen två oföränderliga, publika klassparameterattribut med namnen `nom` för täljaren och `denom` för nämnaren.
- Skapa ett fabriksobjekt med en `apply`-metod som tar två heltalsparametrar och skapar en instans av klassen `Rational`.
- Skapa olika instanser av din klass `Rational` ovan med hjälp av fabriksobjektet.

Uppgift 10. *Skapa en case-klass.* Med en case-klass får man `toString` och fabriksobjekt på köpet. Man behöver inte skriva **val** framför klassparametrar i case-klasser; klassparametrar blir publika, oföränderliga attribut automatiskt när man deklarerar en case-klass.

```
1 scala> case class Pt(x: Double, y: Double)
2 scala> val p = Pt(1.0, 42.0)
3 scala> p.toString
4 scala> println(p)
5 scala> println(Pt(5,6))
```

- Implementera din klass `Rational` från föregående uppgift, men nu som en case-klass.
- Skapa en case-klass `Complex` som representerar komplext tal. Ge klassen två oföränderliga, publika `Double`-attribut: `re` som lagrar realdelen och `im` som lagrar imaginärdelen.

Uppgift 11. *Metoder på datastrukturer.* En datastruktur blir mer användbar om det finns metoder som kan användas på datastrukturen. Metoder i Scala

kan även ha (vissa) specialtecken som namn, t.ex. + enligt nedan.

```
1 scala> case class Point(x: Double, y: Double) {
2     def length: Double = math.hypot(x, y)
3     def add(p: Point): Point = Point(x + p.x, y + p.y)
4     def +(p: Point): Point = Point(x + p.x, y + p.y)
5 }
```

- Använd metoden `length` för att ta reda på vad punkten med koordinaterna (3, 4) har för avstånd till origo?
- Skriv satser som skapar två punkter (3,4) och (5, 6) och låt variablerna `p1` och `p2` referera till respektive punkt. Låt variabeln `p3` bli summan av `p1` och `p2`. Vad får uttrycken `p3.x` resp. `p3.y` för värden?

Uppgift 12. Operatornotation. Vid punktnotation på formen:

`objekt.metod(argument)`

kan man skippa punkten och parenteserna och skriva:

`objekt metod argument`

Detta förenklade skrivsätt kallas **operatornotation**.

- Använd klassen `Point` från uppgift 11 och prova nedan satser. Vilka rader använder operatornotation och vilka rader använder punktnotation? Vilka rader ger felmeddelande?

```
1 scala> val p1 = Point(3,4)
2 scala> val p2 = Point(3,4)
3 scala> p1.add(p2)
4 scala> p1 add p2
5 scala> p1.+(p2)
6 scala> p1 + p2
7 scala> 42 + 1
8 scala> 42.+(1)
9 scala> 42.+ 1
10 scala> 42 +(1)
11 scala> 1.to(42)
12 scala> 1 to 42
13 scala> 1.(to 42)
```

- Implementera metoderna `sub` och `-` i klassen `Point` och skriv uttryck som kombinerar `add` och `sub`, samt `+` och `-` i både punktnotation och operatornotation.
- Operatornotation fungerar även med flera argument. Man använder då parenteser om listan med argumenten: `objekt metod (arg1, arg2)`
Definiera en metod
def `scale(a: Double, b: Double) = Point(x * a, y * b)`
i klassen `Point` och skriv satser som använder metoden med punktnotation och operatornotation.


Uppgift 13. Föränderlighet och oföränderlighet. Oföränderliga och föränderliga objekt beter sig olika vid tilldelning.

- a)  Innan du kör nedan kod: Försök lista ut vad som kommer att skrivas ut. Rita minnessituationen efter varje tilldelning.

```
println("\n--- Example 1: mutable value assignment")
var x1 = 42
var y1 = x1
x1 = x1 + 42
println(x1)
println(y1)
```

- b) Innan du kör nedan kod: Försök lista ut vad som kommer att skrivas ut. Rita minnessituationen efter varje tilldelning. 

```
println("\n--- Example 2: mutable object reference assignment")
class MutableInt(private var i: Int) {
  def +(a: Int): MutableInt = { i = i + a; this }
  override def toString: String = i.toString
}
var x2 = new MutableInt(42)
var y2 = x2
x2 = x2 + 42
println(x2)
println(y2)
```

- c) Innan du kör nedan kod: Försök lista ut vad som kommer att skrivas ut. Rita minnessituationen efter varje tilldelning. 

```
println("\n--- Example 3: immutable object reference assignment")
class ImmutableInt(val i: Int) {
  def +(a: Int): ImmutableInt = new ImmutableInt(i + a)
  override def toString: String = i.toString
}
var x3 = new ImmutableInt(42)
var y3 = x3
x3 = x3 + 42
println(x3)
println(y3)
```

- d) Vad finns det för fördelar med oföränderliga datastrukturer? 

Uppgift 14. Några användbara samlingar. En **samling** (eng. *collection*) är en datastruktur som samlar många objekt av samma typ. I `scala.collection` och `java.util` finns många olika samlingar med en uppsjö användbara metoder. De olika samlingarna i `scala.collection` är ordnade i en gemensam hierarki med många gemensamma metoder; därför har man nytta av det man lär sig om metoderna i en Scala-samling när man använder en annan samling. Vi har redan tidigare sett samlingen `Vector`:

```

1 scala> val tärningskast = Vector.fill(10000)((math.random * 6 + 1).toInt)
2 scala> tä    // tryck TAB
3 scala> tärningskast.    // tryck TAB

```

a) Ungefär hur många metoder finns det som man kan göra på objekt av typen `Vector`? Det är svårt att lära sig alla dessa på en gång, så vi väljer ut några få i kommande uppgifter.

b) Jämför överlappet mellan metoderna i `Vector` och `List` och uppskatta hur stor andel av metoderna som är gemensamma:

```

1 scala> val myntkast =
2   List.fill(10000)(if (math.random < 0.5) "krona" else "klave")
3 scala> my    // tryck TAB
4 scala> myntkast.    // tryck TAB

```

Uppgift 15. Typparameter. Vissa funktioner är generella för många typer och tar en så kallad **typparameter** inom hakparenteser. Ofta slipper man skriva typparametrar, då kompilatorn kan härleda typen utifrån argumenten. Om man anger typparametrar explicit så hjälper kompilatorn dig med att kolla att det verkligen är rätt typ i samlingen.

a) Vad händer nedan?

```

1 scala> var xs = Vector.empty[Int]
2 scala> xs = xs :+ "42"
3 scala> xs = xs :+ 43 :+ 64 :+ 46
4 scala> xs
5 scala> xs := "42".toInt
6 scala> var ys = Vector[Int]("ett", "två", "tre")
7 scala> var ingenting = Vector.empty
8 scala> ingenting = Vector(1,2,3)

```

b) Samlingar är mer användbara om de är *generiska*, vilket innebär att elementens typ avgörs av en typparameter och därför kan vara av vilken typ som helst. Man kan definiera egna funktioner som tar generiska samlingar som parametrar. Förklara vad som händer här:

```

1 scala> val vego = Vector("gurka", "tomat", "apelsin", "banan")
2 scala> val prim = Vector(2, 3, 5, 7, 11, 13)
3 scala> def först[T](xs: Vector[T]): T = xs.head
4 scala> def sist[T](xs: Vector[T]) = xs.last
5 scala> def förstOchSist[T](xs: Vector[T]): (T, T) = (xs.head, xs.last)
6 scala> först(vego)
7 scala> sist(prim)
8 scala> förstOchSist(vego)
9 scala> förstOchSist(prim)
10 scala> def wrap[T](pair: (T, T))(xs: Vector[T]) = pair._1 +: xs :+ pair._2
11 scala> wrap("Odlä", "och ät!")(vego)
12 scala> wrap("Odlä", "och ät!")(vego).mkString(" ")

```

Uppgift 16. Några viktiga samlingsmetoder. Deklarera följande vektorer i REPL.

```

1 scala> val xs = (1 to 10).toVector
2 scala> val a = Vector("abra", "ka", "dabra")
3 scala> val b = Vector("sim", "sala", "bim", "sala", "bim")
4 scala> val stor = Vector.fill(100000)(math.random)

```

Undersök i REPL vad som händer nedan. Alla dessa metoder fungerar på alla samlingar som är indexerbara sekvenser. Givet deklarationerna ovan: vad har uttrycken nedan för värde och typ? Förklara vad som händer hjälp av denna översikt: docs.scala-lang.org/overviews/collections/seqs

- a) `a(1) + xs(1)`
- b) `a apply 0`
- c) `a.isDefinedAt(3)`
- d) `a.isDefinedAt(100)`
- e) `stor.length`
- f) `stor.size`
- g) `stor.min`
- h) `stor.max`
- i) `a indexOf "ka"`
- j) `b.lastIndexOf("sala")`
- k) `"först" ++ b` //minnesregel: colon on the collection side
- l) `a ++ "sist"` //minnesregel: colon on the collection side
- m) `xs.updated(2,42)`
- n) `a.padTo(10, "!")`
- o) `b.sorted`
- p) `b.reverse`
- q) `a.startsWith(Vector("abra", "ka"))`
- r) `"hejsan".endsWith("san")`
- s) `b.distinct`

Uppgift 17. *Några generella samlingsmetoder.* Det finns metoder som går att köra på *alla* samlingar även om de inte är indexerbara. Givet deklarationerna i föregående uppgift: vad har uttrycken nedan för värde och typ? Förklara vad som händer med hjälp av dessa översikter:

docs.scala-lang.org/overviews/collections/trait-traversable

docs.scala-lang.org/overviews/collections/trait-iterable

- a) `a ++ b`
- b) `a ++ stor`
- c) `val ys = xs.map(_ * 5)`
- d) `b.toSet` // En mängd har inga dubletter
- e) `a.head + b.last`
- f) `a.tail`

- g) `a.head ++ a.tail == a`
- h) `Vector(a.head) ++ Vector(b.last)`
- i) `a.take(1) ++ b.takeRight(1)`
- j) `a.drop(2) ++ b.drop(1).dropRight(2)`
- k) `a.drop(100)`
- l) `val e = Vector.empty[String]; e.take(100)`
- m) `Vector(e.isEmpty, e.nonEmpty)`
- n) `a.contains("ka")`
- o) `"ka" contains "a"`
- p) `a.filter(s => s.contains("k"))`
- q) `a.filter(_.contains("k"))`
- r) `a.map(_.toUpperCase).filterNot(_.contains("K"))`
- s) `xs.filter(x => x % 2 == 0)`
- t) `xs.filter(_ % 2 == 0)`

Uppgift 18. De olika samlingarna i `scala.collection` används flitigt i andra paket, exempelvis `scala.util` och `scala.io`.

a) Vad händer här? (Metoden `shuffle` skapar en ny samling med elementen i slumpvis ordning.)

```
1 val xs = Vector(1,2,3)
2 def blandat = scala.util.Random.shuffle(xs)
3 def test = if (xs == blandat) "lika" else "olika"
4 (for(i <- 1 to 100) yield test).count(_ == "lika")
```

b) Skapa en textfil med namnet `fil.txt` som innehåller lite text och läs in den med:

```
scala.io.Source.fromFile("fil.txt", "UTF-8").getLines.toVector
```

```
1 > cat > fil.txt
2 hejsan
3 svejsan
4 > scala
5 scala> val xs = scala.io.Source.fromFile("fil.txt", "UTF-8").getLines.toVector
6 scala> xs.foreach(println)
```

c) Vad händer här? (Metoden `trim` på värden av typen `String` ger en ny sträng med blanktecken i början och slutet borttagna.)

```
1 scala> val pgk =
2   scala.io.Source.fromURL("http://cs.lth.se/pgk/", "UTF-8").getLines.toVector
3 scala> pgk.foreach(println)
4 scala> pgk.map(_.trim).
5   filterNot(_.startsWith("<")).
6   filterNot(_.isEmpty).
7   foreach(println)
```

Uppgift 19. Jämföra List och Vector. En indexerbar sekvens av värden kallas vektor eller lista. I Scala finns flera klasser som kan indexeras, däribland klasserna Vector och List.

a) *Likheter mellan Vector och List.* Kör nedan rader i REPL. Prova indexera i båda och studera hur stor andel av metoderna som är gemensamma.

```
1 scala> val sv = Vector("en", "två", "tre", "fyra")
2 scala> val en = List("one", "two", "three", "four")
3 scala> sv(0) + sv(3)
4 scala> en(0) + en(3)
5 scala> sv. //tryck TAB
6 scala> en. //tryck TAB
```

b) *Skillnader mellan Vector och List.* Klassen Vector i Scala har ”under huven” en avancerad datastruktur i form av ett s.k. självbalanserande träd, vilket gör att Vector är snabbare än List på nästan allt, *utom* att bearbeta elementen i *början* av sekvensen; vill man lägga till och ta bort i början av en List så kan det ibland gå ungefär dubbelt så fort jämfört med Vector, medan alla andra operationer är lika snabba eller snabbare med Vector. Det finns ett fåtal speciella metoder, som bara finns i List, för att skapa en lista och lägga till i början av en lista. Vad händer nedan?

```
1 scala> var xs = "one" :: "two" :: "three" :: "four" :: Nil
2 scala> xs = "zero" :: xs
3 scala> val ys = xs.reverse ::: xs
```

Uppgift 20. Mängd. En mängd är en samling som garanterar att det inte finns några dubletter. Det går dessutom väldigt snabbt, även i stora mängder, att kolla om ett element finns eller inte i mängden. Elementen i samlingen Set hamnar ibland, av effektivitetsskäl, i en förvånande ordning.

```
1 scala> val s = Set("Malmö", "Stockolm", "Göteborg", "Köpenhamn", "Oslo")
2 s: scala.collection.immutable.Set[String] =
3   Set(Oslo, Malmö, Köpenhamn, Stockolm, Göteborg)
4
5 scala> val t = Set("Sverige", "Sverige", "Sverige", "Danmark", "Norge")
6 t: scala.collection.immutable.Set[String] = Set(Sverige, Danmark, Norge)
```

Givet ovan deklarationer: vad blir värde och typ av nedan uttryck?

- a) `s + "Malmö" == s`
- b) `s ++ t`
- c) `Set("Malmö", "Oslo").subsetOf(s)`
- d) `s subsetOf Set("Malmö", "Oslo")`
- e) `s contains "Lund"`
- f) `s apply "Lund"`
- g) `s("Malmö")`
- h) `s - "Stockholm"`

- i) `t - ("Norge", "Danmark", "Tyskland")`
- j) `s -- t`
- k) `s -- Set("Malmö", "Oslo")`
- l) `Set(1,2,3) intersect Set(2,3,4)`
- m) `Set(1,2,3) & Set(2,3,4)`
- n) `Set(1,2,3) union Set(2,3,4)`
- o) `Set(1,2,3) | Set(2,3,4)`

Uppgift 21. *Slå upp värden från nycklar med Map.* Samlingen Map är mycket användbar. Med den kan man snabbt leta upp ett värde om man har en nyckel. Samlingen Map är en generalisering av en vektor, där man kan "indexera", inte bara med ett heltal, utan med vilken typ av värde som helst, t.ex. en sträng. Datastrukturen Map är en s.k. *associativ array*¹, implementerad som en s.k. *hashtabell*².

```
1 scala> var huvudstad =
2   Map("Sverige" -> "Stockholm", "Norge" -> "Oslo", "Skåne" -> "Malmö")
```

Givet ovan variabel huvudstad, förklara vad som händer nedan?

- a) `huvudstad apply "Skåne"`
- b) `huvudstad("Sverige")`
- c) `huvudstad.contains("Skåne")`
- d) `huvudstad.contains("Malmö")`
- e) `huvudstad += "Danmark" -> "Köpenhamn"`
- f) `huvudstad.foreach(println)`
- g) `huvudstad.getOrElse("Norge", "???)`
- h) `huvudstad.getOrElse("Finland", "???)`
- i) `huvudstad.keys.toVector.sorted`
- j) `huvudstad.values.toVector.sorted`
- k) `huvudstad - "Skåne"`
- l) `huvudstad - "Jylland"`
- m) `huvudstad = huvudstad.updated("Skåne", "Lund")`

Uppgift 22. *Skapa Map från en samling.*

- a) `val pairs: Map[String, Long] = Vector(("Björn", 46462229009L), ("Maj", 4`
- b) `val telnr =`
- c)
- d)
- e)
- f)

¹https://en.wikipedia.org/wiki/Associative_array

²https://en.wikipedia.org/wiki/Hash_table

g)

4.4.2 Extrauppgifter: öva mer på grunderna

Uppgift 23. Träna mer på klass

```
class Account(val number: Long, val maxCredit: Int){
  private var balance = 0

  def deposit(amount: Int): Int = {
    if (amount > 0) {balance += amount}
    balance
  }

  def withdraw(amount: Int): (Int, Int) = if (amount > 0) {
    val allowedWithdrawal =
      if (amount < balance + maxCredit) amount
      else balance + maxCredit
    balance = balance - allowedWithdrawal
    (allowedWithdrawal, balance)
  } else (0, balance)

  def show: Unit =
    println("Account Nbr: " + number + " balance: " + balance)
}

object Main {
  def main(args: Array[String]): Unit = {
    ???
  }
}
```

Uppgift 24. Träna mer på mängd

a) Keno-bollar.

4.4.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 25. *Dokumentationen för Any.* Undersök vilka metoder som finns i klassen Any här: <http://www.scala-lang.org/api/current/#scala.Any>. Prova några av metoderna i REPL.

Uppgift 26. *Dokumentationen för samlingar.* Leta upp metoden tabulate i dokumentationen för objektet Vector nästan längst ner i listan här:

[http://www.scala-lang.org/api/current/#scala.collection.immutable.Vector\\$](http://www.scala-lang.org/api/current/#scala.collection.immutable.Vector$)

Leta upp den variant av tabulate som har signaturen:

```
def tabulate[A](n: Int)(f: (Int) => A): Vector[A]
```

Klicka på den gråfyllda trekanten till vänster om signaturen som fäller ut beskrivningen

a) Förklara vad som händer här:

```
scala> Vector.tabulate(10)(i => i % 3)
```

b) Klicka på det blåa stora o-et överst på sidan, för att växla till klass-vyn och studera listan med alla metoder i klassen Vector.

Uppgift 27. *Fler metoder på indexerbara sekvenser.* Deklarera följande vektorer i REPL.

```
1 scala> val xs = (1 to 10).toVector
2 scala> val a = Vector("abra", "ka", "dabra")
3 scala> val b = Vector("sim", "sala", "bim", "sala", "bim")
```

Undersök i REPL vad som händer nedan. Alla dessa metoder fungerar på alla samlingar som är indexerbara sekvenser. Vad har uttrycken för värde och typ? Förklara vad metoden gör. Studera även denna översikt: docs.scala-lang.org/overviews/collections/seqs

- a) `b.indexWhere(s => s.startsWith("b"))`
- b) `a.indices`
- c) `xs.patch(1, Vector(42,43,44), 7)`
- d) `xs.segmentLength(_ < 8, 2)`
- e) `b.sortBy(_.reverse)`
- f) `b.sortWith((s1, s2) => s1.size < s2.size)`
- g) `a.reverseMap(_.size)`
- h) `a intersect Vector("ka", "boom", "pow")`
- i) `a diff Vector("ka")`
- j) `a union Vector("ka", "boom", "pow")`

Uppgift 28. Jämför tidsprestanda mellan List och Vector vid hantering i början och i slutet.

a) Hur snabbt går nedan på din dator? (Exemplet nedan är exekverat på en Intel i7-4790K CPU @ 4.00GHz.)

```
scala> :paste

def time(n: Int)(block: => Unit): Double = {
  def now = System.nanoTime
  var timestamp = now
  var sum = 0L
  var i = 0
  while (i < n) {
    block
    sum = sum + (now - timestamp)
    timestamp = now
  }
}
```

```
    i = i + 1
  }
  val average = sum.toDouble / n
  println("Average time: " + average + " ns")
  average
}

scala> val n = 100000
scala> val l = List.fill(n)(math.random)
scala> val v = Vector.fill(n)(math.random)

scala> (for(i <- 1 to 20) yield time(n){l.take(10)}).min
Average time: 47.1852 ns
Average time: 41.64156 ns
Average time: 105.53986 ns
Average time: 41.91562 ns
Average time: 41.73559 ns
Average time: 63.17134 ns
Average time: 52.93756 ns
Average time: 41.58533 ns
Average time: 41.68017 ns
Average time: 60.18881 ns
Average time: 41.69867 ns
Average time: 41.60771 ns
Average time: 60.32759 ns
Average time: 41.62671 ns
Average time: 43.88916 ns
Average time: 70.47824 ns
Average time: 41.68801 ns
Average time: 41.67223 ns
Average time: 41.67262 ns
Average time: 102.84893 ns
res85: Double = 41.58533

scala> (for(i <- 1 to 20) yield time(n){v.take(10)}).min
Average time: 312.67005 ns
Average time: 88.60023 ns
Average time: 73.21829 ns
Average time: 92.148 ns
Average time: 91.01078 ns
Average time: 87.82874 ns
Average time: 74.04663 ns
Average time: 94.16038 ns
Average time: 88.4243 ns
Average time: 105.88971 ns
Average time: 98.85731 ns
Average time: 72.77369 ns
Average time: 97.04337 ns
Average time: 90.01969 ns
Average time: 88.11196 ns
Average time: 75.20191 ns
Average time: 93.72112 ns
Average time: 110.19777 ns
Average time: 132.4207 ns
Average time: 324.28702 ns
res86: Double = 72.77369
```

```
scala> (for(i <- 1 to 20) yield time(1000){l.takeRight(10)}).min
Average time: 247365.43 ns
Average time: 212801.958 ns
Average time: 212335.938 ns
Average time: 212313.427 ns
Average time: 212524.963 ns
Average time: 219525.627 ns
Average time: 223059.563 ns
Average time: 222426.504 ns
Average time: 221838.828 ns
Average time: 223268.567 ns
Average time: 222739.402 ns
Average time: 222685.229 ns
Average time: 223122.599 ns
Average time: 222683.921 ns
Average time: 222865.865 ns
Average time: 222889.118 ns
Average time: 223247.135 ns
Average time: 222016.82 ns
Average time: 223040.299 ns
Average time: 222624.613 ns
res87: Double = 212313.427

scala> (for(i <- 1 to 20) yield time(1000){v.takeRight(10)}).min
Average time: 2665.715 ns
Average time: 190634.043 ns
Average time: 773.111 ns
Average time: 509.008 ns
Average time: 519.04 ns
Average time: 418.172 ns
Average time: 365.54 ns
Average time: 409.016 ns
Average time: 353.115 ns
Average time: 503.679 ns
Average time: 421.369 ns
Average time: 388.685 ns
Average time: 461.725 ns
Average time: 390.791 ns
Average time: 381.83 ns
Average time: 309.667 ns
Average time: 372.09 ns
Average time: 312.254 ns
Average time: 323.925 ns
Average time: 310.261 ns
res88: Double = 309.667
```

b) Varför går det olika snabbt olika körningar?

Uppgift 29. Studera skillnader i prestanda mellan olika samlingar här:
docs.scala-lang.org/overviews/collections/performance-characteristics.html
(Mer om detta i kommande kurser.)

Uppgift 30. Gör något rekursivt med en lista för att visa hur syntaxen kan se ut med cons.

4.5 Laboration: textfiles

Mål

- ☐ Att lära sig.

Förberedelser

- ☐ Att göra.

4.5.1 Obligatoriska uppgifter

Uppgift 1. En labbuppgiftsbeskrivning.

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

4.5.2 Frivilliga extrauppgifter

Uppgift 2. En labbuppgiftsbeskrivning.

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

Kapitel 5

Sekvensalgoritmer

Koncept du ska lära dig denna vecka:

- | | |
|--|---|
| <input type="checkbox"/> sekvensalgoritm | <input type="checkbox"/> <code>java.util.Scanner</code> |
| <input type="checkbox"/> algoritm: SEQ-COPY | <input type="checkbox"/> <code>java.util.ArrayList</code> |
| <input type="checkbox"/> in-place vs copy | <input type="checkbox"/> <code>scala.collection.mutable.Buffer</code> |
| <input type="checkbox"/> algoritm: SEQ-REVERSE | <input type="checkbox"/> <code>StringBuilder</code> |
| <input type="checkbox"/> algoritm: SEQ-REGISTER | <input type="checkbox"/> <code>java.util.Random</code> |
| <input type="checkbox"/> sekvenser i Java vs Scala | <input type="checkbox"/> slumpvalsfrö |
| <input type="checkbox"/> for-sats i Java | |

5.1 Vad är en sekvensalgoritm?

Ta en **sekvens** som indata och gör något med den!

Två olika sätt:

- Skapa **ny sekvens** utan att förändra indatasekvensen
- Ändra **på plats** (eng. *in place*) i den **föränderliga** indatasekvensen

5.2 Några indexerbara samlingar

- Oföränderliga:
 - Kan **ej** ändra elementreferenserna:
Scala: **Vector**, **List**
- Föränderliga: kan **ändra** elementreferenserna
 - Kan **ej ändra storlek** efter allokering:
Scala+Java: **Array**
 - Kan ändra storlek efter allokering:
Scala: **ArrayBuffer**
Java: **ArrayList**

5.3 Algoritm: SEQ-COPY

Indata : Heltalsarray xs

Resultat: En ny heltalsarray som är en kopia av xs .

```

1  $n \leftarrow$  antalet element i  $xs$ 
2  $ys \leftarrow$  en ny array med plats för  $n$  element
3  $i \leftarrow 0$ 
4 while  $i < n$  do
5    $ys(i) \leftarrow xs(i)$ 
6    $i \leftarrow i + 1$ 
7 end
8 return  $ys$ 
```

5.4 Övning: sequences

Mål



Förberedelser



5.4.1 Grunduppgifter

Uppgift 1. *Variabelt antal argument.* Det går fint att deklarera en funktion som tar en argumentsekvens av godtycklig längd. Syntaxen består av en asterisk * efter typen.

a) Vad händer nedan?

```
1 scala> def printAll(xs: Int*) = xs.foreach(println)
2 scala> printAll(42)
3 scala> printAll(1, 2, 7, 42)
4 scala> def printStrings(wa: String*) = println(wa)
5 scala> printStrings("hej", "på", "dej")
```

b) Vad har parametern wa i printStrings ovan för typ?

c) Ändra i printAll så att även längden på xs skrivs ut före utskriften av alla element. Testa att anropa printAll med olika antal parametrar.


d) Vad händer om du anropar printAll med noll parametrar?

Uppgift 2. *Oföränderliga sekvenser med föränderliga objekt.*

a) Vad får xs för värde efter att attributet i objektet som c2 refererar till ändras på rad 4 nedan? Förklara vad som händer.

```
1 scala> class IntCell(var x: Int){override def toString = "[Int](" + x + ")"}
2 scala> val (c1, c2, c3) = (new IntCell(7), new IntCell(8), new IntCell(9))
3 scala> val xs = Vector(c1, c2, c3)
4 scala> c2.x = 42
5 scala> xs
```

 Rita en bild av minnessituationen efter rad 4 ovan.

 Vad krävs för att allt innehåll i en oföränderlig samling garanterat ska förbli oförändrat?

Uppgift 3. Föränderliga, indexerbara sekvenser: Array och ArrayBuffer

a) Samlingen scala.Array har speciellt stöd i JVM och är extra snabb att allokera och indexera i. Dock kan man inte ändra storleken efter att en Array allokerats. Behöver man mer plats kan man kopiera den till en ny, större array. Koden nedan visar hur det kan gå till.

```

1 scala> val xs = Array(42, 43, 44)
2 scala> val ys = new Array[Int](4) //plats för 4 heltal, från början nollor
3 scala> for (i <- 0 until xs.size){ys(i) = xs(i)}
4 scala> ys(3) = 45

```

Definiera funktionen **def** `copyAppend(xs: Array[Int], x): Array[Int]` som implementerar nedan algoritm, *efter* att du rättat de **två buggarna** i algoritmens while-loop:

Indata : Heltalsarray `xs` och heltalet `x`
Resultat : En ny array som är en kopia av `xs` men med `x` tillagt på slutet som extra element.

```

1  $n \leftarrow$  antalet element i xs
2 ys  $\leftarrow$  en ny array med plats för  $n + 1$  element
3  $i \leftarrow 0$ 
4 while  $i \leq n$  do
5   | ys(i)  $\leftarrow$  xs(i)
6 end
7 ys(n)  $\leftarrow x$ 

```

b) Samlingen `scala.collection.mutable.ArrayBuffer` är inte riktigt lika snabb i alla lägen som `scala.Array` men storleksändring hanteras automatiskt, vilket är en stor fördel då man slipper att själv implementera algoritmer liknande `copyAppend` ovan. Speciellt använder man ofta `ArrayBuffer` om man stegvis vill bygga upp en sekvens. Vad händer nedan?

```

1 scala> val xs = scala.collection.mutable.ArrayBuffer.empty[Int]
2 scala> xs.append(1, 2)
3 scala> while (xs.last < 100) {xs.append(xs.takeRight(2).sum); println(xs)}
4 scala> xs.last
5 scala> xs.length

```

c) Talen i sekvensen som produceras ovan kallas Fibonaccital¹. Hur lång ska en Fibonacci-sekvens vara för att det sista elementet ska komma så nära (men inte över) `Int.MaxValue` som möjligt?

Uppgift 4. Kopiering och uppdatering. Metoder på oföränderliga samlingar skapar nya samlingar istället för att ändra. Därför behöver man inte själv skapa kopior. När en *föränderlig* samling uppdateras på plats, syns denna förändring via alla referenser till samlingen.

```

1 scala> val xs = Vector(1, 2, 3)
2 scala> val ys = xs.toArray
3 scala> ys(1) = 42
4 scala> xs
5 scala> ys
6 scala> val zs = ys.toArray
7 scala> zs(1) = 84
8 scala> xs
9 scala> ys


```


¹sv.wikipedia.org/wiki/Fibonaccital

```
10 scala> zs
```

- Syns uppdateringen av objektet som `ys` refererar till via referensen `xs`? Varför?
- Syns uppdateringen av objektet som `zs` refererar till via referensen `ys`? Varför?
- Syns uppdateringen av objektet som `zs` refererar till via referensen `xs`? Varför?

Uppgift 5. Färdig metod för att skapa kopia av array. Om man inte vill att en uppdatering av en föränderlig samling ska få oönskad påverkan på andra koddelar som refererar till samlingen, behöver man göra kopior av samlingen före uppdatering. Det finns färdiga metoder för kopiering av objekt av typen `Array` i paketet `java.util.Arrays`.

 Studera dokumentationen för metoden `java.util.Arrays.copyOf` här: docs.oracle.com/javase/8/docs/api/java/util/Arrays.html#copyOf-int:A-int- Notera att syntaxen för arrayer i Java är annorlunda: När det står `int[]` i Java så motsvarar det `Array[Int]` i Scala. Vad används den andra parametern till?

 Rita en bild av hur minnet ser ut efter varje tilldelning nedan. Vad har `xs`, `ys` och `zs` för värden efter exekveringen av raderna 1–5 nedan? Varför?


```
1 scala> val xs = Array(1, 2, 3, 4)
2 scala> val ys = xs
3 scala> val zs = java.util.Arrays.copyOf(xs, xs.size - 1)
4 scala> xs(0) = 42
5 scala> zs(0) = 84
6 scala> ys
7 scala> xs
8 scala> zs
```


Uppgift 6. Algoritim: *SEQ-REVERSE-COPY*. Implementera nedan algoritm:

Indata : Heltalsarray `xs` och heltalet `x`

Resultat : En ny heltalsarray med elementen i `xs` i omvänd ordning.

```
1  $n \leftarrow$  antalet element i  $xs$ 
2  $ys \leftarrow$  en ny heltalsarray med plats för  $n$  element
3  $i \leftarrow 0$ 
4 while  $i < n$  do
5    $ys(n - i - 1) \leftarrow xs(i)$ 
6    $i \leftarrow i + 1$ 
7 end
8 return  $ys$ 
```

 Skriv implementation med penna och papper. Använd en **while**-sats på samma sätt som i algoritmen. Prova sedan din implementation på dator och kolla så att den fungerar.

 Skriv implementationen med penna och papper igen, men använd nu istället en **for**-sats som räknar baklänges. Prova sedan din implementation på dator och kolla så att den fungerar.

c) Definiera en funktion i REPL med namnet `reverseCopy` med din implementation i uppgift b.

Uppgift 7. *Algoritim: SEQ-REVERSE.* Strängar av typen `String` är oföränderliga. Vill man ändra i en sträng utan att skapa en ny kopia kan man använda en `StringBuffer` enligt nedan algoritm som vänder bak-och-fram på en sträng.

```

Indata   : En sträng s av typen String
Resultat: En ny sträng av typen String
1 sb ← en ny StringBuilder som innehåller s
2 n ← antalet tecken i s
3 i ← 0
4 for i ← 0 to  $\frac{n}{2} - 1$  do
5   |   temp ← sb(i)
6   |   sb(i) ← sb(n - i - 1)
7   |   sb(n - i - 1) ← temp
8 end
9 return sb omvandlad till en String

```

a) Implementera algoritmen ovan i en funktion med signaturen:

```
def reverseString(s: String): String
```

```


// Kod till facit:
def reverseString(s: String): String = {
  val sb = new StringBuilder(s)
  val n = sb.length
  for (i <- 0 until n / 2) {
    val temp = sb(i)
    sb(i) = sb(n - i - 1)
    sb(n - i - 1) = temp
  }
  sb.toString
}

```

b) Använd din funktion `reverseString` från föregående deluppgift i en ny funktion med signaturen:

```
def isPalindrome(s: String): Boolean
```

som avgör om en sträng är en palindrom.²

c) Man kan med en **while**-sats och indexering direkt i en `String` avgöra om en sträng är en palindrom utan att kopiera den till en `StringBuilder`. Implementera en ny variant av `isPalindrome` som använder denna metod. Skriv först algoritmen på papper i pseudo-kod. 

²sv.wikipedia.org/wiki/Palindrom

```
// Kod till facit:  
def isPalindrome(s: String): Boolean = {  
  val n = s.length  
  var foundDiff = false  
  var i = 0  
  while (i < n/2 && !foundDiff) {  
    foundDiff = s(i) != s(n - i - 1)  
    i += 1  
  }  
  !foundDiff  
}
```

Uppgift 8. Keno-dragningar under ett år -> Registrering...

5.4.2 Extrauppgifter: öva mer på grunderna

5.4.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 9. Studera skillnader och likheter mellan

- a) Array
- b) WrappedArray
- c) ArraySeq

genom att läsa mer om dessa arrayvarianter här:

docs.scala-lang.org/overviews/collections/concrete-mutable-collection-classes

docs.scala-lang.org/overviews/collections/arrays.html

stackoverflow.com/questions/5028551/scala-array-vs-arrayseq

Uppgift 10. Studera vad metoden `java.util.Arrays.deepEquals` gör här:

[Arrays.html#deepEquals-java.lang.Object:A-java.lang.Object:A-](#)

Vad skiljer ovan metod från metoden `java.util.Arrays.equals`?

5.5 Laboration: cardgame

Mål

- ☐ Att lära sig.

Förberedelser

- ☐ Att göra.

5.5.1 Obligatoriska uppgifter

Uppgift 1. En labbuppgiftsbeskrivning.

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

5.5.2 Frivilliga extrauppgifter

Uppgift 2. En labbuppgiftsbeskrivning.

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

Kapitel 6

Klasser, Likhhet

Koncept du ska lära dig denna vecka:

- ☐ objektorientering
- ☐ klass
- ☐ Point
- ☐ Rectangle
- ☐ inkapsling
- ☐ accessregler
- ☐ private
- ☐ public
- ☐ private[this]
- ☐ getters och setters
- ☐ klassparameter
- ☐ primär konstruktor
- ☐ alternativ konstruktor
- ☐ referensvariabler vs enkla värden
- ☐ referenstilldelning vs värdetilldelning
- ☐ referenslikhet vs strukturelikhet
- ☐ eq vs ==
- ☐ compareTo
- ☐ implementera equals

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

Koncept du ska lära dig denna vecka:

- ☐ arv
- ☐ polymorfism
- ☐ likhet vid arv
- ☐ asInstanceOf
- ☐ klasser i Scala vs Java
- ☐ Any vs java.lang.Object
- ☐ klasshierarkin i Scala: Any Any-Ref AnyVal Nothing Null
- ☐ klasshierarkin i Scalas samlingar
- ☐ Shape som basklass till Point och Rectangle
- ☐ accessregler vid arv
- ☐ protected
- ☐ private[this]
- ☐ final
- ☐ abstrakt klass
- ☐ trait
- ☐ inmixning
- ☐ klass vs trait
- ☐ case-object
- ☐ typer med uppräknade värden
- ☐ värdeklasser extends AnyVal
- ☐ implementera equals vid polymorfism ???

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

Koncept du ska lära dig denna vecka:

- ☐ mönstermatchning
- ☐ match
- ☐ Option
- ☐ null
- ☐ try
- ☐ catch

- ☐ Try
- ☐ unapply
- ☐ flatten
- ☐ flatMap
- ☐ partiella funktioner
- ☐ collect

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, Typparametrar

Koncept du ska lära dig denna vecka:

- | | |
|---|---|
| <input type="checkbox"/> matris | <input type="checkbox"/> generisk funktion |
| <input type="checkbox"/> nästlade for-satser | <input type="checkbox"/> generisk klass |
| <input type="checkbox"/> designexempel: Tre-i-rad | <input type="checkbox"/> matriser i Java vs Scala |

9.1 Övning: matrices

Mål



Förberedelser



9.1.1 Grunduppgifter

Uppgift 1.

```
1 scala> class Cell[T](var x: T){  
2     val typeName: String = x.getClass.getTypeName  
3     override def toString = "[" + typeName + "]" + x + "  
4 }
```

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.

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

9.2.2 Frivilliga extrauppgifter

Uppgift 2. En labbuppgiftsbeskrivning.

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

Kapitel 10

Sökning, Sortering

Koncept du ska lära dig denna vecka:

- | | |
|---|---|
| <input type="checkbox"/> algoritm: LINEAR-SEARCH | <input type="checkbox"/> algoritm: INSERTION-SORT |
| <input type="checkbox"/> algoritm: BINARY-SEARCH | <input type="checkbox"/> algoritm: SELECTION-SORT |
| <input type="checkbox"/> algoritmisk komplexitet | <input type="checkbox"/> mer om filer |
| <input type="checkbox"/> sortering till ny vektor | <input type="checkbox"/> serialisering |
| <input type="checkbox"/> sortering på plats | |

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

Koncept du ska lära dig denna vecka:

- | | |
|---|--|
| <input type="checkbox"/> skillnader mellan Scala och Java | <input type="checkbox"/> primitiva typer i Java |
| <input type="checkbox"/> for-sats i Java | <input type="checkbox"/> wrapperklasser i Java |
| <input type="checkbox"/> java for-each i Java | <input type="checkbox"/> samlingar i Java vs Scala |
| <input type="checkbox"/> autoboxing i Java | <input type="checkbox"/> scala.collection.JavaConverters |
| | <input type="checkbox"/> enum 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.

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

11.2.2 Frivilliga extrauppgifter

Uppgift 2. En labbuppgiftsbeskrivning.

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

Kapitel 12

Trådar, Web ???, Android ???

Koncept du ska lära dig denna vecka:

- | | |
|-------------------------------------|---|
| <input type="checkbox"/> Thread | <input type="checkbox"/> (Javascript ???) |
| <input type="checkbox"/> Future | <input type="checkbox"/> (css ???) |
| <input type="checkbox"/> Duration | <input type="checkbox"/> Scala.js ??? |
| <input type="checkbox"/> Await | <input type="checkbox"/> Android ??? |
| <input type="checkbox"/> (HTML ???) | |

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.

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

12.2.2 Frivilliga extrauppgifter

Uppgift 2. En labbuppgiftsbeskrivning.

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

Kapitel 13

Design

Koncept du ska lära dig denna vecka:



Kapitel 14

Tentaträning

Koncept du ska lära dig denna vecka:



Del III

Appendix

Appendix A

Virtuell maskin

A.1 Vad är en virtuell maskin?

Du kan köra alla kursens verktyg i en så kallad virtuell maskin (vm). Det är ett enkelt och säkert sätt att installera ett nytt operativsystem i en "sandlåda" som inte påverkar din dators ursprungliga operativsystem.

A.2 Installera kursens vm

Det finns en virtuell maskin förberedd med alla verktyg som du behöver förinstallerade. Gör så här:

1. Installera VirtualBox v5 här:
<https://www.virtualbox.org/wiki/Downloads>
2. Ladda ner filen vbox.zip här:
<http://fileadmin.cs.lth.se/pgk/vbox.zip>
OBS! Då filen är på nästan 4GB kan nedladdningen ta mycket lång tid.
3. Packa upp filen vbox.zip i biblioteket "VirtualBox VMs" som du fick i din hemkatalog när du installerade VirtualBox. Du får då 3 filer som heter något med "introprog-ubuntu-64bit".
4. Kolla med hjälp av denna sida:
<https://md5file.com/calculator>
så att filen "introprog-ubuntu-64bit.vdi" har denna sha256-checksumma:
— ska-stå-checksumma-här-sen —
5. Öppna VirtualBox och lägg till maskinen introprog-ubuntu-64bit genom menyn "add".
6. Starta maskinen.
7. Öppna ett terminalfönster och skriv `scala` och du är igång och kan göra första övningen!

A.3 Vad innehåller kursens vm?

Den virtuella maskinen kör Xubuntu 14.04 med fönstermiljön XFCE, vilket är samma miljö som E-husets linuxdatorer kör.

I den virtuella maskinen finns detta förinstallerat:

- Java JDK 8
- Scala 2.11.8
- Kojo 2.4.08
- Eclipse Mars.2 med ScalaIDE 4.3
- gedit med syntaxfärgning för Scala och Java
- git
- sbt
- Ammonite REPL

Appendix B

Terminalfönster och kommandoskal

B.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.

B.2 Några viktiga terminalkommando

Tipsa om ss64.com

Appendix C

Editera

C.1 Vad är en editor?

C.2 Välj editor

Appendix D

Kompilera och exekvera

D.1 Vad är en kompilator?

D.2 Java JDK

D.2.1 Installera Java JDK

D.3 Scala

D.3.1 Installera Scala-kompilatorn

D.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 effekten. En sådan tolk kallas Read-Evaluate-Print-Loop eftersom den läser en rad i taget och översätter till maskinkod som körs direkt.

D.4.1 Scala REPL

Kommandon i REPL

`:paste`

Kortkommandon: Ctrl+K etc.

Appendix E

Dokumentation

E.1 Vad gör ett dokumentationsverktyg?

E.2 scaladoc

E.3 javadoc

Appendix F

Integrerad utvecklingsmiljö

F.1 Vad är en IDE?

F.2 Kojo

F.2.1 Installera Kojo

www.kogics.net/kojo-download

F.2.2 Använda Kojo

Tabell F.1: Några av sköldpaddans funktioner. Se även 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/

F.3 Eclipse och ScalaIDE

F.3.1 Installera Eclipse och ScalaIDE

F.3.2 Använda Eclipse och ScalaIDE

Appendix G

Byggverktyg

G.1 Vad gör ett byggverktyg?

G.2 Byggverktyget sbt

G.2.1 Installera sbt

G.2.2 Använda sbt

Appendix H

Versionshantering och kodlagring

H.1 Vad är versionshantering?

H.2 Versionshanteringsverktyget git

H.2.1 Installera git

H.2.2 Använda git

H.3 Vad är nyttan med en kodlagringsplats?

H.4 Kodlagringsplatsen GitHub

H.4.1 Installera klienten för GitHub

H.4.2 Använda GitHub

H.5 Kodlagringsplatsen Atlassian BitBucket

H.5.1 Installera SourceTree

H.5.2 Använda SourceTree

Appendix I

Nyckelord

I.1 Vad är ett nyckelord ord?

Nyckelord är ord i ett programmeringsspråk som har speciell betydelse och reserverade för endast ett användningsområde. Nyckelord kallas även *reserverade ord*¹. Man kan till exempel inte använda nyckelordet **def** som namn på en variabel. Nyckelord ges ofta en speciell färg av de keditorer som erbjuder *syntaxstyrd färgning*.

I.2 Nyckelord i Scala

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

I.3 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.

abstract continue for new switch

¹Läs mer här: en.wikipedia.org/wiki/Reserved_word

```
assert *** default goto * package synchronized
boolean do if private this
break double implements protected throw
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 J

Lösningsförslag till övningar

J.1 expressions

J.1.1 Grunduppgifter

Uppgift 1.

- a) 43
- b) Lösningstext.

J.1.2 Extrauppgifter: öva mer på grunderna

Uppgift 2.

- a) 42
- b) Lösningstext.

J.1.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 3.

- a) 42
- b) Lösningstext.

J.2 programs

J.2.1 Grunduppgifter

Uppgift 1.

- a) 42
- b) Lösningstext.

Uppgift 2.

J.2.2 Extrauppgifter: öva mer på grunderna

Uppgift 3.

- a) 42
- b) Lösningstext.

J.2.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 4.

- a) 42
- b) Lösningstext.

J.3 functions

J.3.1 Grunduppgifter

Uppgift 1.

- a) 42
- b) Lösningstext.

J.3.2 Extrauppgifter: öva mer på grunderna

Uppgift 2.

- a) 42
- b) Lösningstext.

J.3.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 3.

- a) 42
- b) Lösningstext.

J.4 data

J.4.1 Grunduppgifter

Uppgift 1.

- a) 42
- b) Lösningstext.

J.4.2 Extrauppgifter: öva mer på grunderna

Uppgift 2.

- a) 42
- b) Lösningstext.

J.4.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 3.

- a) 42
- b) Lösningstext.

J.5 sequences

J.5.1 Grunduppgifter

Uppgift 1.

- a) 42
- b) Lösningstext.

J.5.2 Extrauppgifter: öva mer på grunderna

Uppgift 2.

- a) 42
- b) Lösningstext.

J.5.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 3.

- a) 42
- b) Lösningstext.

J.6 classes

J.6.1 Grunduppgifter

Uppgift 1.

- a) 42
- b) Lösningstext.

J.6.2 Extrauppgifter: öva mer på grunderna

Uppgift 2.

- a) 42
- b) Lösningstext.

J.6.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 3.

- a) 42
- b) Lösningstext.

J.7 traits

J.7.1 Grunduppgifter

Uppgift 1.

- a) 42
- b) Lösningstext.

J.7.2 Extrauppgifter: öva mer på grunderna

Uppgift 2.

- a) 42
- b) Lösningstext.

J.7.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 3.

- a) 42
- b) Lösningstext.

J.8 matching

J.8.1 Grunduppgifter

Uppgift 1.

- a) 42
- b) Lösningstext.

J.8.2 Extrauppgifter: öva mer på grunderna

Uppgift 2.

- a) 42
- b) Lösningstext.

J.8.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 3.

- a) 42
- b) Lösningstext.

J.9 matrices

J.9.1 Grunduppgifter

Uppgift 1.

- a) 42
- b) Lösningstext.

J.9.2 Extrauppgifter: öva mer på grunderna

Uppgift 2.

- a) 42
- b) Lösningstext.

J.9.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 3.

- a) 42
- b) Lösningstext.

J.10 sorting

J.10.1 Grunduppgifter

Uppgift 1.

- a) 42
- b) Lösningstext.

J.10.2 Extrauppgifter: öva mer på grunderna

Uppgift 2.

- a) 42
- b) Lösningstext.

J.10.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 3.

- a) 42
- b) Lösningstext.

J.11 `scalajava`

J.11.1 Grunduppgifter

Uppgift 1.

- a) 42
- b) Lösningstext.

J.11.2 Extrauppgifter: öva mer på grunderna

Uppgift 2.

- a) 42
- b) Lösningstext.

J.11.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 3.

- a) 42
- b) Lösningstext.

J.12 threads

J.12.1 Grunduppgifter

Uppgift 1.

- a) 42
- b) Lösningstext.

J.12.2 Extrauppgifter: öva mer på grunderna

Uppgift 2.

- a) 42
- b) Lösningstext.

J.12.3 Fördjupningsuppgifter: avancerad nivå

Uppgift 3.

- a) 42
- b) Lösningstext.

Appendix K

Ordlista