Programmering, grundkurs

Övningar

Datavetenskap, LTH
Lunds Universitet

Kompilerad: 26 augusti 2016

http://cs.lth.se/pgk

Innehåll

1. 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 deklarera 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 interval.

Förberedelser

- ☐ Studera begreppen i kapitel **??**.
- ☐ Du behöver en dator med Scala installerad, se appendix **??**.

1.1 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 2.11.8 (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 (eller tryck pil-upp) men ”glöm bort” att skriva högerparentesen innan du trycker på *Enter*. Vad händer?

c) Evaulera 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 * 4` (byt ev. ut `0`:an mot siffran efter `res` i utskriften från förra evalueringen). Vad har uttrycket för värde och typ?

```
scala> res0 * 4
```

Uppgift 2. Vad är en *literal*?
en.wikipedia.org/wiki/Literal_(computer programming)

Uppgift 3. Vilken typ har följande literaler? Försök först gissa vilken typen blir; testa sedan i REPL och notera vad det blev för typ.

a) `15`
b) `32L`
c) `'*'`
d) `"*"`
e) `42.0`
f) `84D`
g) `32d`
h) `23F`
i) `18f`
j) `true`
k) `false`

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

```
scala> def p = { print("hej"); println("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? Försök först gissa vilket värde och vilken typ det blir; testa sedan i REPL och notera resultatet.

a) `1 + 41`
b) `1.0 + 18`
c) `42.toDouble`
d) `(41 + 1).toDouble`
e) `1.042e42`
f) `12E6.toLong`

g) `"gurk" + 'a'`

h) `'A'`

i) `'A'.toInt`

j) `'0'.toInt`

k) `'1'.toInt`

l) `'9'.toInt`

m) `'A' + '0'`

n) `('A' + '0').toChar`

o) `"*!%#".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) `9L + 3d`

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

a) `23 + 2 * 2`

b) `(23 + 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 uttrycken i deluppgifterna a till h nedan?

a) `42 / 2`

b) `42 / 4`

c) `42.0 / 4`

d) `1 / 4`

e) `1 % 4`

f) `45 % 42`

g) `42 % 2`

h) `41 % 2`

i) Skriv ett uttryck som ”plockar ut” siffran `7` ur talet `5793` med hjälp av heltalsdivision och moduloräkning.

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`. Genom att trycka på tab tagenten kan man se vad som finns i olika paket.

```
scala> java.    //tryck TAB efter punkten
applet   awt   beans   io   lang   math   net   nio   rmi   security   sql

scala>
```

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?

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

b) Undersök dokumentationen för klassen `java.lang.Math` här:
https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html
Vad gör `java.lang.Math.hypot`?

c) Undersök dokumentationen för paketobjektet `scala.math` här:
http://www.scala-lang.org/api/current/#scala.math.package
Ge exempel på någon funktion i `java.lang.Math` som inte finns i `scala.math`.

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

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`

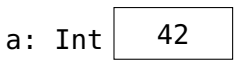
e) `true || true`

f) `false || true`
g) `true || false`
h) `false || false`
i) `42 == 42`
j) `42 != 42`
k) `42.0001 == 42`
l) `42.0000000000000001 == 42`
m) `42.0001 > 42`
n) `42.0000000000000001 > 42`
o) `42.0001 >= 42`
p) `42.0000000000000001 <= 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.


```
scala> var a = 13
scala> var b = a + 1
scala> var c = (a + b) * 2.0
scala> b = 0
scala> a = 0
scala> c = c + 1
```

Efter första raden ser minnessituationen ut så här:



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

```
scala> var x = 30
scala> x + 1
scala> x
scala> x = x + 1
scala> x
scala> x == x + 1
scala> val y = 20
scala> y = y + 1
scala> var z = {println("gurka"); 10}
scala> def w = {println("gurka"); 10}
scala> z
scala> z
scala> z = z + 1
```

```
scala> w
scala> w
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`**?

✎ d) Tilldela variabeln **`val`** `even` värdet av ett uttryck som med modulo-operatorn `%` och likhetsoperatorn == testar om ett tal `n` är jämnt.

✎ e) Tilldela variabeln **`val`** `odd` värdet av ett uttryck som med modulo-operatorn `%` och olikhetsoperatorn `!=` testar om ett tal `n` är udda.

✎ **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.

```
scala> var a = 40
scala> var b = a + 40
scala> a += 10
scala> b -= 10
scala> a *= 2
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, på ett typsäkert sätt, infoga variabelvärden i en sträng. Variablernas namn ska föregås med ett dollartecken, t.ex. `s"Hej $namn"`. Om man vill evaluera ett uttryck placeras detta inom klammer direkt efter dollartecknet, t.ex. `s"Dubbla längden: ${namn.size * 2}"`

```
scala> val f = "Kim"
scala> val e = "Robinsson"
scala> val tot = f.size + e.size
scala> println(s"Namnet '$f $e' har $tot bokstäver.")
scala> println(s"Efternamnet '$e' har ${e.size} bokstäver.")
```

a) Vad skrivs ut ovan?

b) Skapa följande utskrifter med hjälp av stränginterpolatorn `s` och lämpliga variabler.

```
Namnet 'Kim' har 3 bokstäver.
Namnet 'Robinsson' har 9 bokstäver.
```

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

```
scala> if (true) println("sant") else println("falskt")
scala> if (false) println("sant") else println("falskt")
```

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

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*. Med bakåtpilen `<-` kan man i en `for`-sats ange vilka värden som ska gås igenom i sekvens. Vid varje runda i loopen får en lokal variabel ett nytt värde i sekvensen.

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

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

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

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

Uppgift 21. Repetition med metoden `foreach`. Efter framåtpilen `=>` (se nedan) anges vad som ska hända för varje element som gås igenom sekventiellt. Vid varje runda i loopen får en lokal variabel ett nytt värde i sekvensen.

a) Vad ger nedan satser för utskrifter?

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

b) Använd `foreach` och skriv ut följande:

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

Uppgift 22. `while`*-sats*. En sats eller ett block med satser upprepas så länge ett villkor är sant.

a) Vad ger nedan satser för utskrifter?

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

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

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

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

Uppgift 23. *Slumptal*. Undersök vad dokumentationen säger om funktionen `scala.math.random`:
http://www.scala-lang.org/api/current/#scala.math.package

✎ a) Vilken typ har värdet som returneras av funktionen `random`?

✎ b) Vilket är det minsta respektive största värde som kan returneras?

✎ c) Är `random` en *äkta* funktion (eng. *pure function*) i matematisk mening?

d) Anropa funktionen `math.random` upprepade gånger och notera vad som händer. Använd pil-upp-tangenten.

```
scala> math.random
```

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

```
scala> for (i <- 1 to 20) println((math.random * 3 + 1).toInt)
```

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 samma rad.

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

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

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

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

j) Förklara vad som händer nedan.

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

Uppgift 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`

1.2 Extrauppgifter

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`.)

Uppgift 26. Undersök vad som finns i paketet `scala.math` genom att studera dess dokumentation: www.scala-lang.org/api/current/#scala.math.package och gör några matematiska beräkningar i REPL som använder olika funktioner i `math`-paketet.

Uppgift 27. Antag att du byter plats mellan satsen efter villkoret och satsen efter **`else`** i **`if`**-satsen nedan. Hur kan du ändra i villkoret så att det ändå skrivs ut samma sak som före bytet?

```
if (x == 42) println("the meaning of it all") else println(":(")
```

Uppgift 28. Rita en ny bild av datorns minne efter varje evaluerad rad nedan. Bilderna ska visa variablers namn, typ och värde.

```
1 scala> var x = 42
2 scala> var y = x + 1
3 scala> x += -1
4 scala> y -= 1
```

Uppgift 29. Skapa med hjälp av **`while`** några olika oändliga loopar som skriver ut olika saker vid varje loop-runda.

Uppgift 30. Hitta på några egna övningar för att träna mer på De Morgans lagar.

1.3 Fördjupningsuppgifter

Uppgift 31. Läs om moduloräkning här en.wikipedia.org/wiki/Modulo_operation och undersök hur det blir med olika tecken (positivt resp. negativt) på divisor och dividend.

Uppgift 32. Läs om identifierare i Scala och speciellt *literal identifiers* här: `http://www.artima.com/pins1ed/functional-objects.html#6.10`.

a) Förklara vad som händer nedan:

```
scala> val `konstig val` = 42
scala> println(`konstig val`)
```

b) Scala och Java har olika uppsättningar med reserverade ord. På vilket sätt kan ”backticks” vara använbart med anledning av detta?

Uppgift 33. Sök upp dokumentationen för `java.lang.Integer`.

a) Undersök i REPL hur metoderna `toBinaryString` och `toHexString` fungerar.

b) Vad betyder literalen `0x2a`?

Uppgift 34. Typannoteringar skapas genom att i ett uttryck placera ett kolon följt av en typ, vid behov omslutet av en parentes. Skapa ett större uttryck med typannoteringar och försök få kompilatorn att kontrollera typen på intressanta ställen. Märk att typannoteringar också ibland kan användas för att konvertera mellan numeriska typer.

Uppgift 35. Förklara vad som händer nedan:

```
1 scala> var i = 42
2 scala> i += 1
3 scala> i *= 2
4 scala> i /= 3
```

Uppgift 36. Läs om BigInt och BigDecimal här: alvinalexander.com/scala/how-to-use-large-integer-decimal-numbers-in-scala-bigint-bigdecimal och prova att skapa riktigt stora tal med hjälp av metoden `pow` på BigInt och tal med riktigt många decimaler med BigDecimal dess metod `pow`.

Uppgift 37. Sök upp dokumentationtionen för `java.lang.Math.multiplyExact` och läs om vad den metoden gör.

a) Vad händer här?

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

b) Varför kan man vilja använda `java.lang.Math.multiplyExact` i stället för ”vanlig” multiplikation?

c) Sök med Ctrl+F i webbläsaren och efter förekomster av texten *”overflow”* i javadoc för klassen `java.lang.Math` i JDK 8. Vad är ”overflow”? Vilka metoder finns i `java.lang.Math` som hjälper dig att upptäcka om det blir overflow?

Uppgift 38. 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 39. För typerna `Byte`, `Short`, `Char`, `Int`, `Long`, `Float`, `Double`: Undersök hur många bitar som behövs för att representera varje typs omfång?
Tips: Några användbara uttryck:
`Integer.toBinaryString(Int.MaxValue + 1).size`
`Integer.toBinaryString((math.pow(2,16) - 1).toInt).size`
`1 + 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 40. 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
```

2. 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 begreppen i kapitel **??**.
- ☐ Bekanta dig med grundläggande terminalkommandon, se appendix **??**.
- ☐ Bekanta dig med den editor du vill använda, se appendix **??**.

2.1 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`

r) Datastrukturen `Range` håller reda på start- och slutvärde, samt stegstorleken för en uppräkning, men alla talen i uppräkningen genereras inte förrän så behövs. En `Int` tar 4 bytes i minnet. Ungefär hur mycket plats i minnet tar de objekt som variablerna `r` respektive `a` refererar till nedan?

```
scala> val r = (1 to Int.MaxValue by 2)
scala> val a = r.toArray
```

Tips: Använd uttrycket `BigInt(Int.MaxValue) * 2` i dina beräkningar.

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. *Algoritm: 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:

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

Uppgift 8. *Skript.* Skapa en fil med namn `hello-script.scala` med hjälp av en editor 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.

```
> scala hello-script.scala
hej skript
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.

```
> scala hello-script.scala 5001
hej skript
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 ← 0
2 i ← 1
3 while i ≤ n do
4 |   sum ← 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 två buggar i algoritmen. Vilka? Rätta buggarna och test igen genom att ”köra” algoritmen med penna på papper och kontrollera så att algoritmen fungerar för $n = 0$, $n = 1$, och $n = 5$. Vad händer om $n = -1$?

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 standarfunktioner 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. *Algoritm: 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?

a) `val a = {1 + 1; 2 + 2; 3 + 3; 4 + 4}; println(a)`

b) `val b = {1; 2; 3; {val b = 4; b + b; b + 1}}; println(b)`

c) `{val a = 42; println(a)}`

d) `{val a = 42}; println(a)`

e) `{val a = 42; {val a = 43; println(a)}; println(a)}`

f) `{var a = 42; {a = a + 1}; var a = 43}`

g) `{var a = 42; {a = a + b; var b = 43}; println(a)}`

h) `{var a = 42; {var b = 43; a = a + b}; println(a)}`

i) `{var a = 42; {a = a + b; def b = 43}; println(a)}`

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

k)

```
{
  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"))
```

c) 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 {
```

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

```
      def hello = println("Hej paket p1.p12!")
    }
  }
}

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
  }
}
```

```
> gedit paket.scala
> scalac paket.scala
> scala gurka.tomat.banan.Main
> ls -R
```

Uppgift 15. *Skapa `jar`-filer och använda classpath*

a) 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?

b) 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.

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

Uppgift 16. *Skapa dokumentation med `scaladoc`-kommandot*

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

```
> scaladoc paket.scala
> ls
> 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 paketdokumentationskommentar p2 */
package p2 {
  /** min paketdokumentationskommentar 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)
    }
  }
}
```

```
> gedit paket.scala
> scalac paket.scala
> jar cvf mittpaket.jar gurka
> scala -cp mittpaket.jar
scala> gurka.tomat.banan.p2.p21.hello.date
scala> :q
> scaladoc paket.scala
> firefox index.html
```

2.2 Extrauppgifter

Uppgift 17. Implementera algoritmen MININDEX som söker index för minsta heltalet i en sekvens. Pseudokod för algoritmen MININDEX:

```
   Indata : Sekvens xs med n st heltal.
   Utdata : Index för det minsta talet eller −1 om xs är tom.
 1 minPos ← 0
 2 i ← 1
 3 while i < n do
 4 |   if xs(i) < xs(minPos) then
 5 |   |   minPos ← i
 6 |   end
 7 |   i ← i + 1
 8 end
 9 if n > 0 then
10 |   return minPos
11 else
12 |   return −1
13 end
```

a) Prova algoritmen med penna och papper på sekvensen $(1, 2, -1, 4)$ och rita minnessituationen efter varje runda i loopen. Vad blir skillnaden i exekveringsförloppet om loopvariablen i initialiserats till 0 i stället för 1?

b) Implementera algoritmen MININDEX i Scala i en funktion med denna signatur:

```
def indexOfMin(xs: Array[Int]): Int = ???
```

Testa för olika fall: tom sekvens; sekvens med endast ett tal; lång sekvens med det minsta talet först, någonstans mitt i, samt sist.

```
// kod till facit
def indexOfMin(xs: Array[Int]): Int = {
  var minPos = 0
  var i = 1
  while (i < xs.size) {
    if (xs(i) < xs(minPos)) minPos = i
    i += 1
  }
  if (xs.size > 0) minPos else -1
}
```

2.3 Fördjupningsuppgifter

Uppgift 18. Läs om krullparenteser och vanliga parenteser på stack overflow: stackoverflow.com/questions/4386127/what-is-the-formal-difference-in-scala-between-braces-and-parentheses-and-when och prova själv i REPL hur du kan blanda dessa olika slags parenteser på olika vis.

Uppgift 19. Gör jämförande studier av Scalas api-dokumentation för `ArrayBuffer`, `Array` och `Vector`. Ge exempel på metoder som finns på objekt av typen `Array` och `ArrayBuffer` men inte på objekt av typen `Vector`. *Tips:* Kolla efter metoder som returnerar `Unit`. Prova några muterande metoder på `Array` och `ArrayBuffer` i REPL.

Uppgift 20. Bygg vidare på koden nedan och gör ett Sten-Sax-Påse-spel[2] 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 %-operatorn för att avgöra vem som vinner.

```
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
    }
  }
}
```

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

3. 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 skapa och använda anonyma funktioner (s.k. lambda-funktioner).
- ☐ 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.

Förberedelser

- ☐ Studera begreppen i kapitel **??**.

3.1 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.

```
scala> ???  // definiera funktionen öka
scala> öka(42)
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, sum(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?

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

Uppgift 4. *Funktionsanrop med namngivna argument.*

```
scala> def skrivNamn(förnamn: String, efternamn: String) =
         println("Namn: " + efternamn + ", " + förnamn)
scala> skrivNamn("Kim", "Robinson")
scala> skrivNamn(förnamn = "Viktor", efternamn = "Oval")
scaka> 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:

```
scala> def görDetta(block: => Unit) = block
scala> görDetta(println("hej"))
scala> görDetta{println("goddag")}
scala> görDetta{println("hej"); println("svejs")}
scala> def görDettaTvåGånger(block: => Unit) = {block; block}
scala> görDettaTvåGånger{println("goddag")}
```

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

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

Uppgift 9. *Skapa din egen kontrollstruktur.*

a) Använd fördröjd evaluering i kombination med en uppdelad parameterlista och skapa din egen kontrollstruktur enligt nedan. (Det är så här som loopen `upprepa` i Kojo är definierad.)

```
scala> def upprepa(n: Int)(block: => Unit): Unit = {
         var i = 0
         while (i < n) {block; i = i + 1}
       }
```

b) Använd din nya loop-procedur och förklara vad som händer nedan.

```
scala> upprepa(10)(println("hej"))
scala> upprepa(1000){
  val tärning = (math.random * 6 + 1).toInt
  print(tärning + " ")
}
```

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

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

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

```
scala> def applicera(x: Int, f: Int => Int) = f(x)
scala> def dubbla(x: Int) = 2 * x
scala> def halva(x: Int) = x / 2
scala> val d = applicera(42, dubbla)
scala> val h = applicera(42, halva)
```

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

```
scala> def sum(a: Int)(b: Int) = a + b
scala> sum(1)(2)
scala> val f = sum(42) _
scala> f(1)
scala> val inc = sum(1) _
scala> val dec = sum(-1) _
scala> inc(42)
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 (så som vi är vana vid inom matematiken).[3]

```
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 är objektets 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:

```
scala> object plus { def apply(x: Int, y: Int) = x + y }
scala> plus.apply(42,43)
scala> plus(42, 43)
scala> val add: (Int, Int) => Int = (x, y) => x + y
scala> add(42, 42)
scala> add.   // tryck på TAB
scala> add.apply(42, 42)
scala> val inc = add.curried(1)
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 slumpmässigt heltal i intervallet $[a, b]$. Anropa objektets `apply`-metod med `(1 to 100).foreach(i => print(??? + " "))` för att skriva ut 100 slumptal mellan 1 och 6. Prova både att explicit anropa `apply` med punktnotation och att använda funktionsappliceringssyntax.

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

a) Förklara vad som händer här:

[3] Äkta funktioner uppfyller per definition *referentiell transparens* (eng. *referential transparency*) som du kan läsa mer om här: en.wikipedia.org/wiki/Referential_transparency

```
scala> val olat = 42
scala> lazy val lat = 42
scala> println(lat)
scala> val nu = {Thread.sleep(1000); println("nu"); 42}
scala> lazy val sen = {Thread.sleep(1000); println("sen"); 42}
scala> def igen = {Thread.sleep(1000); println("hver gang"); 42}
scala> println(nu)
scala> println(sen)
scala> println(igen)
scala> println(nu)
scala> println(sen)
scala> println(igen)
scala> object m {lazy val stor = Array.fill(1e9.toInt)(liten); val liten = 42}
scala> m.liten
scala> m.stor
```

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

c) Förklara vad som händer här:

```
scala> object objektÄrLata { val sen = { println("nu!"); 42 } }
scala> objektÄrLata
scala> objektÄrLata.sen
scala> {val x = y; val y = 42}
scala> object buggig {val a = b; val b = 42}
scala> buggig.a
scala> object funkar {lazy val a = b; val b = 42}
scala> funkar.a
scala> object nowarning {val many = Array.fill(10)(one); val one = 1}
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.

✎ b) 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.

✎ b) 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 *funktionsliteraler* 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 funktionsliteralerna 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)
```

3.2 Extrauppgifter

Uppgift 20. Skriv och testa en funktion `avg` som räknar ut medelvärdet mellan två heltal och returnerar en `Double`.

Uppgift 21. Skriv och testa nedan två varianter av beräkning av avståndet mellan två punkter:

```
scala> def dist(x1: Int, y1: Int, x2: Int, y2: Int): Double = ???
scala> def dist(p1: (Int, Int), p2: (Int, Int)): Double = ???
```

TODO!!! Fler förslag på extrauppgifter om funktioner välkomna, speciellt på det som man kan behöva öva mer på för att lättare förstå!

3.3 Fördjupningsuppgifter

Uppgift 22. *Undersök den genererade byte-koden.* Kompilatorn genererar **byte-kod**, uttalas ”bajtkod” (eng. *byte code*), som den virtuella maskinen tolkar och översätter till maskinkod medan programmet kör. Med kommandot `:javap` i REPL kan du undersöka byte-koden.

```
scala> def plusxy(x: Int, y: Int) = x + y
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 byte-koden med `:javap plusxyz`. Vad skiljer byte-koden mellan `plusxy` och `plusxyz`?

c) Läs om byte-kod här: en.wikipedia.org/wiki/Java_bytecode. Vad betyder den inledande bokstaven i additionsinstruktionen?

Uppgift 23. *Undersök svansrekursion genom att kasta undantag.* Förklara vad som händer. Kan du hitta bevis för att kompilatorn kan optimera rekursionen till en vanlig loop?

```
scala> def explode = throw new Exception("BANG!!!")
scala> explode
scala> lastException.printStackTrace
scala> def countdown(n: Int): Unit =
         if (n == 0) explode else countdown(n-1)
scala> countdown(10)
scala> lastException.printStackTrace
scala> def countdown2(n: Int): Unit =
         if (n == 0) explode else {countdown2(n-1); print("no tailrec")}
scala> countdown2(10)
scala> countdown2(1000)
scala> lastException
scala> lastException.getStackTrace.size
```

```
scala> :javap countdown
scala> :javap countdown2
```

Uppgift 24. *`@tailrec`-annotering.* Du kan be kompilatorn att ge felmeddelande om den inte kan optimera koden till motsvarande en while-loop. Om den inte kan det hämmas prestanda och det finns risk för en överfull anropssstack (eng. *stack overflow*). Prova nedan rader i REPL och förklara vad som händer.

```
scala> def countNoTailrec(n: Long): Unit =
         if (n <= 0L) println("Klar! " + n) else {countNoTailrec(n-1L); ()}
scala> countNoTailrec(1000L)
scala> countNoTailrec(100000L)
scala> import scala.annotation.tailrec
scala> @tailrec def countNoTailrec(n: Long): Unit =
         if (n <= 0L) println("Klar! " + n) else {countNoTailrec(n-1L); ()}
scala> @tailrec def countTailrec(n: Long): Unit =
         if (n <= 0L) println("Klar! " + n) else countTailrec(n-1L)
scala> countTailrec(1000L)
scala> countTailrec(100000L)
scala> countTailrec(Int.MaxValue.toLong * 2L)
```

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 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 begreppen i kapitel **??**.

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 understreck följt av ordningsnumret.

```
scala> val namn = ("Pippi", "Långstrump")
scala> namn._1
scala> namn._2
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 bankonto. Instanser av typen `Konto` håller reda på hur mycket pengar det finns på kontot och vilket kontonumret är. Datavärden som sparas i varje objektinstans, så som `saldo` och `nummer`, kallas **attribut** (eng. *attribute*) eller **fält** (eng. *field*).

```
scala> class Konto {
         var saldo = 0
         var nummer = 0L
       }
scala> val k1 = new Konto
scala> val k2 = new Konto
scala> k1.saldo = 1000
scala> k1.nummer = 12345L
scala> k2.saldo = 2000
scala> k2.nummer = 67890L
scala> println("Konto: " + k1.nummer + " Saldo:" + k1.saldo)
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 skapas med `new`, kan man placera attributen i en parameterlista till klassen. Koden som körs när objektet skapas och attributen tilldelas sina initialvärden, kallas **konstruktor** (eng. *constructor*).

```
scala> class Konto(var saldo: Int, val nummer: Long)
scala> val k = new Konto(0, 12345L)
scala> println("Konto: " + k.nummer + " Saldo:" + k.saldo)
scala> println(k)
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 eller privat attribut?* Man kan förhindra att ett attribut syns utanför klassen med hjälp av nyckelordet **`private`**.

```
scala> class Konto1(val nummer: Long){ var saldo = 0 }
scala> val k1 = new Konto1(12345678901L)
scala> k1.nummer
scala> k1.saldo += 1000
scala> class Konto2(val nummer: Long){ private var saldo = 0 }
scala> val k2 = new Konto2(12345678901L)
scala> k2.nummer
scala> k2.saldo += 1000
```

a) Vad händer ovan?

b) 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
  }
}
```

c) Spara koden i en fil, kompilera med `scalac` 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`.

d) 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.

e) Förbättra metoderna `in` och `ut` så att man inte kan sätta in eller ta ut negativa belopp.

f) 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?

```
scala> class Punkt(val x: Double, val y: Double)
scala> val pt: Punkt = new Punkt(10.0, 10.0)
scala> val i: Int = pt.x
scala> val (x: Double, y: Double) = (pt.x, pt.y)
scala> val p: Double = new Punkt(5.0, 5.0)
scala> val p = new Punkt(5.0, 5.0): Double
scala> val p = new Punkt(5.0, 5.0): Punkt
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?

```
scala> class Punkt(val x: Double, val y: Double)
scala> val pt: Punkt = new Punkt(1.0, 2.0)
scala> pt.isInstanceOf[Punkt]
scala> pt.isInstanceOf[Double]
scala> pt.x.isInstanceOf[Punkt]
scala> pt.x.isInstanceOf[Double]
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?

```
scala> class Punkt(val x: Double, val y: Double)
scala> val pt: Punkt = new Punkt(1.0, 2.0)
scala> pt.isInstanceOf[Punkt]
scala> pt.isInstanceOf[Any]
scala> pt.x.toString
scala> println(pt.x)
scala> val a: Any = pt
scala> println(a.x)
scala> a.toString
scala> pt.y.toString
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.

```
scala> class Punkt(val x: Double, val y: Double) {
         override def toString: String = "[x=" + x + ",y=" + y + "]"
```

```
      }
scala> val pt = new Punkt(1.0, 42.0)
scala> pt.toString
scala> println(pt)
```

a) Vad händer egentligen på sista raden ovan?

b) Omdefiniera toString så att den ger en sträng på formen `Punkt(1.0, 42.0)`.

c) 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)
}
```

a) Skriv satser som använder metoden `apply` i fabriksobjektet **`object`** `Pt` för att skapa flera olika punkter.

b) Ge applymetoden default-argument 0.0 för både x och y så att `Pt()` skapar en punkt i origo.

c) 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.

d) Skapa ett fabriksobjekt med en `apply`-metod som tar två heltalsparametrar och skapar en instans av klassen `Rational`.

e) 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.

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

a) Implementera din klass `Rational` från föregående uppgift, men nu som en case-klass.

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.

```
scala> case class Point(x: Double, y: Double) {
         def distToOrigin: Double = math.hypot(x, y)
         def add(p: Point): Point = Point(x + p.x, y + p.y)
         def +(p: Point): Point = add(p)
       }
```

a) Använd metoden `distToOrigin` för att ta reda på vad punkten med koordinaterna (3, 4) har för avstånd till origo?

b) 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 med hjälp av metoden `add`. 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**.

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

```
scala> val p1 = Point(3,4)
scala> val p2 = Point(3,4)
scala> p1.add(p2)
scala> p1 add p2
scala> p1.+(p2)
scala> p1 + p2
scala> 42 + 1
scala> 42.+(1)
scala> 42.+ 1
scala> 42 +(1)
scala> 1.to(42)
scala> 1 to 42
scala> 1.to(42)
```

b) Implementera metoderna `sub` och `-` i klassen `Point` och skriv uttryck som kombinerar add och sub, samt + och - i både punktnotation och operatornotation.

c) 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 assigmnent")
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`:

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

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

```
scala> var xs = Vector.empty[Int]
scala> xs = xs :+ "42"
scala> xs = xs :+ 43 :+ 64 :+ 46
scala> xs
scala> xs :+= "42".toInt
scala> var ys = Vector[Int]("ett", "två", "tre")
scala> var ingenting = Vector.empty
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:

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

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

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

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

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

Uppgift 20. *Mängd.* En mängd är en samling som garanterar att det inte finns några dubbletter. Det går dessutom väligt 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.

```
scala> val s = Set("Malmö", "Stockholm", "Göteborg", "Köpenhamn", "Oslo")
s: scala.collection.immutable.Set[String] =
     Set(Oslo, Malmö, Köpenhamn, Stockholm, Göteborg)

scala> val t = Set("Sverige", "Sverige", "Sverige", "Danmark", "Norge")
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*[4], implementerad som en s.k. *hashtabell*[5].

```
scala> var huvudstad =
  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) Definiera denna vektor och undersök dess typ:

```
val pairs = Vector(
  ("Björn", 46462229009L),
  ("Maj", 46462221667L),
  ("Gustav", 46462224906L))
```

[4] https://en.wikipedia.org/wiki/Associative_array
[5] https://en.wikipedia.org/wiki/Hash_table

b) Vad har variablen `telnr` nedan för typ:

```
var telnr = pairs.toMap
```

c) Använd `telnr` för att slå upp telefonnummer för Maj och Kim med hjälp av metoderna `apply` och `get`.

d) Använd metoden `getOrElse` vid upplagningar av `telnr` och ge `-1L` som telefonnummer i händelse av att ett nummer inte finns.

e) Lägg till `("Fröken Ur", 464690510L)` i `telnr`-mappen.

f) Skapa en `Vector[(String, String)]` enligt nedan, så att telefonnumret blir en sträng utan inledande landsnummer men med en nolla i riktnumret. Byt ut `???` mot lämpligt uttryck.

```
1 scala> telnr.toVector.map(p => ???)
2 res85: Vector[(String, String)] = Vector(("Björn", "0462229009"), ("Maj",
3 "0462221667"), ("Gustav", "0462224906"), ("Fröken Ur", 04690510"))
```

g) Använd vektorn i resultatet ovan för att skapa en ny `Map[String, String]` med nationella telefonnumer. Slå upp numret till Fröken Ur.

Uppgift 23. *Samlingsmetoden `maxBy`.* Med samlingsmetoden `maxBy` kan man själv definiera vad som ska maximeras. (Denna metod kommer du att behöva i veckans laboration.)

a) Förklara vad som händer nedan.

```
1 scala> val xs = Vector((2,3), (1,5), (-1, 1), (7, 2))
2 scala> xs.maxBy(x => x._1)
3 scala> xs.maxBy(x => x._2)
```

b) Om man bara använder en parameter i en anonym funktion, till exempel parametern `x` i lambdauttrycket `x => x + 1` *en enda* gång, och kompilatorn kan gissa alla typer, kan man använda understreck som ”platshållare” för att förkorta lambdauttrycket så här: `_ + 1`

Skriv uttrycken på raderna 2 och 3 i föregående deluppgift på ett kortare sätt med hjälp platshållarsyntax (eng. *place holder syntax*).

c) På motsvarande sätt kan man använda `minBy` för att välja vilket funktion som definierar minimum. Prova `minBy` på motsvarande sätt som i föregående deluppgifter.

Uppgift 24. *Samlingsmetoden `sliding`.* I veckans labb kommer du att ha nytta av samlingsmetoden `sliding`, som ger en iterator för speciella delsekvenser av en sekvens, vilka kan liknas vid ”utsikten” i ett ”glidande fönster”. Kör nedan i REPL och beskriv vad som händer.

```
1 scala> val xs = Vector("fem", "gurkor", "är", "fler", "än", "fyra", "tomater")
2 scala> xs.sliding(2).toVector
3 scala> xs.sliding(3).toVector
4 scala> xs.sliding(4).toVector
5 scala> xs.sliding(7).toVector
6 scala> xs.sliding(10).toVector
```

4.2 Extrauppgifter

Uppgift 25. Skriv nedan program med en editor och kompilera från terminalen. Lägg till kod i huvudprogrammet som testar klassen `Account` och kompilera och kör. Utvidga sedan klassen `Account` med fler attribut och funktioner som du väljer själv.

```
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 26. Läs om reglerna för spelet Keno här:
https://sv.wikipedia.org/wiki/Keno och gör deluppgifterna nedan.

a) Skapa en klass `Keno` som kan användas för att genomföra en Kenodragning. Låt klassen ha ett privat attribut `balls` som är en föränderlig mängd med heltal och som från början är tom. Implementera lämpliga metoder i klassen för att användaren av klassen ska kunna dra nya slumpmässiga bollar som inte redan är dragna.

b) Skapa en `case class KenoBet(bet: Set[Int])` för att hålla reda vilka 11 bollar en viss person satsar på. Definiera en metod
`def numberOfHits(keno: Keno): Int = ???`
i case-klassen `KenoBet` som givet en kenodragning räknar ut hur många bollar som satsats rätt.

c) Skriv ett huvudprogram som simulerar en enkel Kenodragning. Låt två personer satsa på 11 slumpmässiga bollar, genomför en dragning av 20 bollar ur 70 möjliga och kontrollera sedan hur många bollar som personerna har prickat rätt.

4.3 Fördjupningsuppgifter

Uppgift 27. *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 28. *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$
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 29. *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 30. 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
Welcome to Scala 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_66).
Type in expressions for evaluation. Or try :help.

scala> :pa
// Entering paste mode (ctrl-D to finish)

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
}

// Exiting paste mode, now interpreting.

time: (n: Int)(block: => Unit)Double

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: 476.85004 ns
Average time: 52.29291 ns
Average time: 221.50289 ns
Average time: 218.60302 ns
Average time: 45.01888 ns
Average time: 243.7818 ns
Average time: 45.02228 ns
Average time: 2132.03995 ns
Average time: 52.83995 ns
Average time: 46.7478 ns
Average time: 51.8753 ns
Average time: 70.57658 ns
Average time: 45.26142 ns
Average time: 95.16307 ns
Average time: 43.84092 ns
Average time: 55.24695 ns
```

```
Average time: 84.06113 ns
Average time: 42.04872 ns
Average time: 50.9871 ns
Average time: 122.80649 ns
res0: Double = 42.04872


scala> (for(i <- 1 to 20) yield time(n){v.take(10)}).min
Average time: 429.23201 ns
Average time: 257.70543 ns
Average time: 271.02261 ns
Average time: 198.8826 ns
Average time: 161.67466 ns
Average time: 190.5253 ns
Average time: 112.82044 ns
Average time: 82.45798 ns
Average time: 81.17192 ns
Average time: 129.28968 ns
Average time: 104.86973 ns
Average time: 80.33942 ns
Average time: 81.64533 ns
Average time: 92.22053 ns
Average time: 78.5791 ns
Average time: 84.55555 ns
Average time: 78.88382 ns
Average time: 77.25284 ns
Average time: 83.62473 ns
Average time: 72.39703 ns
res1: Double = 72.39703

scala> (for(i <- 1 to 20) yield time(1000){l.takeRight(10)}).min
Average time: 264902.261 ns
Average time: 225706.676 ns
Average time: 228625.873 ns
Average time: 230358.379 ns
Average time: 229971.679 ns
Average time: 237404.948 ns
Average time: 242580.96 ns
Average time: 242455.325 ns
Average time: 242316.002 ns
Average time: 242046.311 ns
Average time: 242378.896 ns
Average time: 242740.221 ns
Average time: 242131.301 ns
Average time: 242466.169 ns
Average time: 242075.599 ns
Average time: 242247.534 ns
Average time: 242739.886 ns
Average time: 241982.93 ns
Average time: 242118.373 ns
Average time: 241941.998 ns
res2: Double = 225706.676

scala> (for(i <- 1 to 20) yield time(1000){v.takeRight(10)}).min
Average time: 661.737 ns
Average time: 420.765 ns
```

```
Average time: 225.867 ns
Average time: 256.524 ns
Average time: 235.596 ns
Average time: 231.764 ns
Average time: 154.279 ns
Average time: 139.37 ns
Average time: 139.183 ns
Average time: 153.957 ns
Average time: 142.883 ns
Average time: 140.837 ns
Average time: 154.178 ns
Average time: 138.72 ns
Average time: 202.93 ns
Average time: 174.179 ns
Average time: 175.98 ns
Average time: 171.658 ns
Average time: 177.097 ns
Average time: 173.1 ns
res3: Double = 138.72
```

b) Varför går det olika snabbt olika körningar?

Uppgift 31. Studera skillnader i prestanda mellan olika samlingar här: docs.scala-lang.org/overviews/collections/performance-characteristics.html (Mer om detta i kommande kurser.)

Uppgift 32. För samlingen `List` finns en alternativ metod till `+:` som heter `::` och kallas ”cons” och som i kombination med objektet `Nil` kan användas för att med alternativ syntax bygga listor. Läs om detta här: alvinalexander.com/scala/how-create-scala-list-range-fill-tabulate-constructors och hitta på några egna övningar för att undersöka hur cons och Nil fungerar. Metoder som slutar med kolon är högerassociativa. Läs mer om detta här: http://www.artima.com/pins1ed/basic-types-and-operations.html#5.8

5. Övning: sequences

Mål

- ☐ Kunna implementera funktioner som tar argumentsekvenser av godtycklig längd.
- ☐ Kunna tolka enkla sekvensalgoritmer i pseudokod och implementera dem i programkod, t.ex. tillägg i slutet, insättning, borttagning, omvändning, etc., både genom kopiering till ny sekvens och genom förändring på plats i befintlig sekvens.
- ☐ Kunna använda föränderliga och oföränderliga sekvenser.
- ☐ Förstå skillnaden mellan om sekvenser är föränderliga och om innehållet i sekvenser är föränderligt.
- ☐ Kunna välja när det är lämpligt att använda `Vector`, `Array` och `ArrayBuffer`.
- ☐ Känna till att klassen `Array` har färdiga metoder för kopiering.
- ☐ Kunna implementera algoritmer som registrerar antalet förekomster av objekt i en sekvens som indexeras med antalet förekomster.
- ☐ Kunna generera sekvenser av pseudoslumptal med specificerat slumptalsfrö.
- ☐ Kunna implementera sekvensalgoritmer i Java med `for`-sats och primitiva arrayer.
- ☐ Kunna beskriva skillnaden i syntax mellan arrayer i Scala och Java.
- ☐ Kunna använda klassen `java.util.Scanner` i Scala och Java för att läsa in heltalssekvenser från `System.in`.

Förberedelser

- ☐ Studera begreppen i kapitel **??**.

5.1 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
```

✎ b) Rita en bild av minnessituationen efter rad 4 ovan.

✎ c) 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: Int): Array[Int]` som implementerar nedan algoritm, *efter* att du rätta de **två buggarna** i algoritmens while-loop:

Indata : Heltalsarray xs och heltalet x
Resultat : En ny array som 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 $\quad ys(i) \leftarrow xs(i)$
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, 1)
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 Fibonaccital[6]. 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.

```
scala> val xs = Vector(1, 2, 3)
scala> val ys = xs.toArray
scala> ys(1) = 42
scala> xs
scala> ys
scala> val zs = ys.toArray
scala> zs(1) = 84
scala> xs
scala> ys
scala> zs
```

a) Syns uppdateringen av objektet som `ys` refererar till via referensen `xs`? Varför?

b) Syns uppdateringen av objektet som `zs` refererar till via referensen `ys`? Varför?

c) 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`.

a) 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?

b) 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?

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

[6] sv.wikipedia.org/wiki/Fibonaccital

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

Indata : Heltalsarray xs
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 $\quad ys(n-i-1) \leftarrow xs(i)$
6 $\quad i \leftarrow i+1$
7 **end**
8 **return** ys

✎ a) 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.

✎ b) 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. *Algoritm: SEQ-REVERSE.* Strängar av typen `String` är oföränderliga. Vill man ändra i en sträng utan att skapa en ny kopia kan man använda en `StringBuilder` 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 $sb \leftarrow$ en ny `StringBuilder` som innehåller s
2 $n \leftarrow$ antalet tecken i s
3 **for** $i \leftarrow 0$ **to** $\frac{n}{2} - 1$ **do**
4 $\quad temp \leftarrow sb(i)$
5 $\quad sb(i) \leftarrow sb(n-i-1)$
6 $\quad sb(n-i-1) \leftarrow temp$
7 **end**
8 **return** sb omvandlad till en `String`

a) Implementera algoritmen ovan i en funktion med signaturen:

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

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.[7]

✎ 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`.

[7] sv.wikipedia.org/wiki/Palindrom

Implementera en ny variant av `isPalindrome` som använder denna metod. Skriv först algoritmen på papper i pseudo-kod.

Uppgift 8. *Algoritm: SEQ-REGISTER.* Algoritmer för registrering löser problemet att räkna förekomst av olika saker, till exempel antalet tärningskast som gav en sexa. Antag att vi har följande vektor `xs` som representerar 13 st tärningskast:

```
scala> val xs =  Vector(5, 3, 1, 6, 1, 3, 5, 1, 1, 6, 3, 2, 6)
```

a) Använd metoderna `filter` och `size` på `xs` för att filtrera ut alla 6:or och räkna hur många de är.

b) Använd metoderna `filter` och `size` på `xs` för att filtrera ut alla jämna kast och räkna hur många de är.

c) Metoden `groupBy` på en samling tar en funktion `f` som parameter och skapar en ny `Map` med nycklar `k` som är associerade till samlingar som utgör grupper av värden där `f(x) == k`. Vad händer här:

```
scala> xs.groupBy(x => x % 2)
scala> xs.groupBy(_ % 2)
scala> xs.groupBy(_ % 3)
scala> xs.groupBy(_ % 3).foreach(println)
scala> val freqEvenOdd = xs.groupBy(_ % 2).map(p => (p._1, p._2.size))
scala> val nEven = freqEvenOdd(0)
scala> val nOdd = freqEvenOdd(1)
```

d) Använd metoden `groupBy` på `xs` med den s.k. identitetsfunktionen `i => i` som returnerar sitt eget argument. Vad händer?

e) Definiera en `val freq: Map[Int, Int]` som räknar antalet olika tärningsutfall i `xs`. Använd metoden `groupBy` på `xs` med identitetsfunktionen följt av en `map` med funktionen `p => (p._1, p._2.size)`.

f) Du ska nu själv implementera en registreringsalgoritm. Skriv en funktion:

```
def tärningsRegistrering(xs: Array[Int]): Array[Int] = ???
```

som implementerar nedan algoritm (som alltså inte använder `groupBy` eller andra färdiga metoder på samlingar förutom `size` och `apply`).

Indata : En array xs med heltal mellan 1 och 6 som representerar utfall av många tärningskast.
Resultat : En array f med 7 st element där $f(0)$ innehåller totala antalet kast, $f(1)$ anger antalet ettor, $f(2)$ antalet tvåor, etc. till och med $f(6)$ som anger antalet sexor.

$f \leftarrow$ en ny array med 7 element där alla element initaliseras till 0.
$f(0) \leftarrow$ antalet element i xs
$i \leftarrow 0$
while $i < f(0)$ **do**
    $f(xs(i)) \leftarrow f(xs(i)) + 1$
    $i \leftarrow i + 1$
end
return f

Testa din funktion med nedan funktionsanrop:

```
scala> tärningsRegistrering(Array.fill(1000)((math.random * 6).toInt +1))
res12: Array[Int] = Array(1000, 174, 174, 167, 171, 145, 169)
```

Uppgift 9. *Algoritm: SEQ-REMOVE-COPY.* Ibland vill man kopiera alla element till en ny `Array` *utom* ett element på en viss plats `pos`.

a) Skriv algoritmen SEQ-REMOVE-COPY i pseudokod med penna på papper.

b) Implementera algoritmen SEQ-REMOVE-COPY i en funktion med denna signatur:

```
def removeCopy(xs: Array[Int], pos: Int): Array[Int]
```

Uppgift 10. *Algoritm: SEQ-REMOVE.* Ibland vill man ta bort ett element på en viss position i en befintlig `Array` utan att kopiera alla element till en ny `Array`. Ett sätt att göra detta är att flytta alla efterföljande element ett steg mot lägre index och låta sista platsen bli 0.

a) Skriv algoritmen SEQ-REMOVE i pseudokod med penna på papper.

b) Implementera algoritmen SEQ-REMOVE i en funktion med denna signatur:

```
def remove(xs: Array[Int], pos: Int): Unit
```

Uppgift 11. *Deterministiska pseudoslumptalssekvenser med `java.util.Random`.* Klassen `java.util.Random` ger möjlighet att generera en sekvens av tal som verkar slumpmässiga. Genom att välja ett visst s.k. **frö** (eng. *seed*) kan man få samma sekvens av pseudoslumptal varje gång.

a) Sök upp och studera dokumentationen för `java.util.Random`. Hur skapar man en ny instans av klassen `Random`? Vad gör operationen `nextInt` på `Random`-

objekt.

b) Förklara vad som händer nedan?

```
scala> import java.util.Random
scala> val frö = 42L
scala> val rnd = new Random(frö)
scala> rnd.nextInt(10)
scala> (1 to 100).foreach(_ => print(rnd.nextInt(10)))
scala> val rnd1 = new Random(frö)
scala> val rnd2 = new Random(frö)
scala> val rnd3 = new Random(System.nanoTime)
scala> val rnd4 = new Random((math.random * Long.MaxValue).toLong)
scala> def flip(r: Random) = if (r.nextInt(2) > 0) "krona" else "klave"
scala> val xs = (1 to 100).map{i =>
     (flip(rnd1), flip(rnd2), flip(rnd3), flip(rnd4))}
scala> xs foreach println
scala> xs.exists(q => q._1 != q._2)
scala> xs.exists(q => q._1 != q._3)
```

c) Nämn några sammanhang då det är användbart att kunna bestämma fröet till en slumptalssekvens.

d) Blir det samma sekvens om du använder fröet `42L` som argument till konstruktorn vid skapandet av en instans av `java.util.Random` på en *annan* dator?

e) Sök reda på dokumentationen för `java.math.random` och undersök hur denna sekvens skapas.

f) Vad blir det för frö till slumptalssekvensen om man skapar ett `Random`-objekt med hjälp av konstruktorn utan parameter?

Uppgift 12. *Undersök om tärningskast är rektangelfördelade.*[8]
Skriv en funktion `def testRandom(r: Random, n: Int): Unit = ???` som ger följande utskrift:

```
scala> val rnd = new Random(42L)
scala> testRandom(rnd, 1000)
Antal kast: 1000
Antal 1:or: 178
Antal 2:or: 187
Antal 3:or: 167
Antal 4:or: 148
Antal 5:or: 155
Antal 6:or: 165
```

Tips: Anropa din funktion `tärningsRegistrering` från uppgift 8.

Uppgift 13. *Array och `for`-sats i Java.*

[8] För ett rektangelfördelat slumpvärde gäller att om man drar (nästan oändligt många) slumpvärden så blir det (nästan) lika många av varje möjligt värde. Om man ritar en sådan fördelning i ett koordinatsystem med antalet utfall på y-axeln och de olika värdena på x-axeln, så blir bilden rektangelformad. Du får lära dig mer om sannolikhetsfördelningar i kommande kurser i matematisk statistik.

a) Skriv nedan program i en editor och spara i filen `DiceReg.java`:

```
// DiceReg.java
import java.util.Random;

public class DiceReg {
    public static void main(String[] args) {
        int[] diceReg = new int[6];
        int n = 100;
        Random rnd = new Random();
        if (args.length > 0) {
            n = Integer.parseInt(args[0]);
        }
        System.out.print("Rolling the dice " + n + " times");
        if (args.length > 1) {
            int seed = Integer.parseInt(args[1]);
            rnd.setSeed(seed);
            System.out.print(" with seed " + seed);
        }
        System.out.println(".");
        for (int i = 0; i < n; i++) {
            int pips = rnd.nextInt(6);
            diceReg[pips]++;
        }
        for (int i = 1; i <= 6; i++) {
            System.out.println("Number of " + i + "'s: " +
                diceReg[i-1]);
        }
    }
}
```

b) Kompilera med `javac DiceReg.java` och kör med `java DiceReg 10000 42` och förklara vad som händer.

c) Beskriv skillnaderna mellan Scala och Java, vad gäller syntaxen för array och `for`-sats. Beskriv några andra skillnader mellan språken som syns i programmet ovan.

d) Ändra i programmet ovan så att loop-variabeln `i` skrivs ut i varje runda i varje `for`-sats. Kompilera om och kör.

e) Skriv om programmet ovan genom att abstrahera huvudprogrammets delar till de statiska metoderna `parseArguments`, `registerPips` och `printReg` enligt nedan skelett. Notera speciellt hur `private` och `public` är angivet. Spara programmet i filen `DiceReg2.java` och kompilera med `javac DiceReg2.java` i terminalen.

```
// DiceReg2.java
import java.util.Random;
```

```
public class DiceReg2 {
    public static int[] diceReg = new int[6];
    private static Random rnd = new Random();

    public static int parseArguments(String[] args) {
        // ???
        return n;
    }

    public static void registerPips(int n){
        // ???
    }

    public static void printReg() {
        // ???
    }

    public static void main(String[] args) {
        int n = parseArguments(args);
        registerPips(n);
        printReg();
    }
}
```

f) Starta Scala REPL i samma bibliotek som filen `DiceReg2.class` ligger i och kör nedan satser och förklara vad som händer:

```
scala> DiceReg2.main(Array("1000","42"))
scala> DiceReg2.diceReg
scala> DiceReg2.registerPips(1000)
scala> DiceReg2.printReg
scala> DiceReg2.registerPips(1000)
scala> DiceReg2.printReg
scala> DiceReg2.rnd
```

g) Växla synligheten på attributen mellan **`private`** och **`public`**, kompilera om och studera effekten i Scala REPL. Hur lyder felmeddelandet om du försöker komma åt en privat medlem?

h) Ange en viktig anledning till att man kan vilja göra medlemmar privata. 

Uppgift 14. *Läsa in tal med `java.util.Scanner`.* Med **`new`** `Scanner(System.in)` skapas ett objekt som kan läsa in tal som användaren skriver i terminalfönstret.

a) Sök upp och studera dokumentationen för `java.util.Scanner`. Vad gör metoderna `hasNextInt()` och `nextInt()`?

b) Skriv nedan program i en editor och spara i filen `DiceScanBuggy.java`:

```
// DiceScanBuggy.java
```

```
import java.util.Random;
import java.util.Scanner;

public class DiceScanBuggy {
    public static int[] diceReg = new int[6];
    public static Scanner scan = new Scanner(System.in);

    public static void registerPips(){
        System.out.println("Enter pips separated by blanks.");
        System.out.println("End with -1 and <Enter>.");
        boolean isPips = true;
        while (isPips && scan.hasNextInt()) {
            int pips = scan.nextInt();
            if (pips >= 1 && pips <=6 ) {
              diceReg[pips]++;
            } else {
              isPips = false;
            }
        }
    }

    public static void printReg() {
        for (int i = 0; i < 6; i++) {
            System.out.println("Number of " + i + "'s: " +
                diceReg[i-1]);
        }
    }

    public static void main(String[] args) {
        registerPips();
        printReg();
    }
}
```

c) Kompilera och kör med indatasekvensen `1 2 3 4 -1` och notera hur registreringen sker.

d) Programmet fungerar inte som det ska. Du behöver korrigera 3 saker för att programmet ska göra rätt. Rätta buggarna och spara det rättade programmet som `DiceScan.java`. Kompilera och testa att det rättade programmer fungerar med olika indata.

✎ **Uppgift 15.** *Välja sekvenssamling.* Vilken av `Vector`, `Array` och `ArrayBuffer` hade du valt i dessa situationer?

a) Ditt program innehåller en sekvens av objekt med data om alla ca 10^7 medborgare i sverige. Efter noggranna mätningar visar det sig att tillägg av objekt på godtyckliga ställen i sekvensen är en flaskhals.

b) Ditt program innehåller en sekvens av objekt med data om ca 10^2 residensstäder i Sverige. Senast det skedde en uppdatering av mängden referensstäder var 1997. Prestandamätningar visar att det är uppdatering av attributvärden i objekten som tar mest tid. Städerna behöver kunna bearbetas i godtycklig ordning.

c) Ditt program innehåller en sekvens av ca 10^9 osorterade heltal som ska läsas in från fil och sorteras på plats i minnet. Det första talet i filen anger antalet heltal. Det är viktigt att sorteringen går snabbt. När talen är sorterade ska de skrivas tillbaka till fil i sorterad ordning.

d) Ditt program innehåller en sekvens av ett känt antal oföränderliga objekt med data om genomförda banktransaktioner. Sekvensen ska bearbetas parallellt i godtycklig ordning med olika algoritmer som kan köras oberoende av varandra.

5.2 Extrauppgifter

Uppgift 16. *Algoritm: SEQ-INSERT-COPY.*

Indata : En sekvens xs av typen `Array[Int]` och heltalen x och pos
Resultat : En ny sekvens av typen `Array[Int]` som är en kopia av xs men där x är infogat på plats pos

1 $n \leftarrow$ antalet element xs
2 $ys \leftarrow$ en ny `Array[Int]` med plats för $n+1$ element
3 **for** $i \leftarrow 0$ **to** $pos - 1$ **do**
4 $\quad ys(i) \leftarrow xs(i)$
5 **end**
6 $ys(pos) \leftarrow x$
7 **for** $i \leftarrow pos$ **to** $n - 1$ **do**
8 $\quad ys(i+1) \leftarrow xs(i)$
9 **end**
10 **return** ys

a) Implementera ovan algoritm i en funktion med denna signatur:

```
def insertCopy(xs: Array[Int], x: Int, pos: Int): Array[Int]
```

b) Vad måste `pos` vara för att det ska fungera med en tom array som argument?

c) Vad händer om din funktion anropas med ett negativt argument för `pos`?

d) Vad händer om din funktion anropas med `pos` lika med `xs.size`?

e) Vad händer om din funktion anropas med `pos` större än `xs.size`?

Uppgift 17. *Algoritm: SEQ-INSERT.* Man kan implementera algoritmen SEQ-INSERT på plats i en `Array[Int]` så att alla elementen efter `pos` flyttas fram ett steg och att sista elementet ”försvinner”.

a) Skriv algoritment SEQ-INSERT i pseudokod med penna och papper.

b) Implemtera SEQ-INSERT i en funktion med denna signatur:

```
def insert(xs: Array[Int], x: Int, pos: Int): Unit
```

Uppgift 18. Implementera funktionen `tärningsRegistrering` från uppgift 8 på nytt, men nu med en `for`-sats istället.

Uppgift 19. Bygg vidare på Keno-uppgiften nummer 26 i kapitel **??** på sidan 51 och gör registrering av det slumpmässiga utfallet av 365 Keno-dragningar och skriv ut frekvenserna för förekomsten av varje boll. Öka sedan antalet dragningar och undersök hur många dragningar du behöver göra för att frekvenserna ska bli nästan lika?

5.3 Fördjupningsuppgifter

Uppgift 20. Sök reda på dokumentationen för metoden `patch` på klassen `Array`.

a) Använd metoden `patch` för att implementera SEQ-INSERT-COPY:

```
def insertCopy(xs: Array[Int], x: Int, pos: Int): Array[Int] =
  xs.patch(???, ???, ???)
```

b) Använd metoden `patch` för att implementera SEQ-REMOVE-COPY:

```
def removeCopy(xs: Array[Int], pos: Int): Array[Int] =
  xs.patch(???, ???, ???)
```

Uppgift 21. 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 22. 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`?

Uppgift 23. *Använda `jline` istället för `Scanner` i REPL.* Om du använder `java.util.Scanner` i Scala REPL så ekas inte de tecken som skrivs, så som sker om du använder scannern med `System.in` i en kompilerad applikation. Om du vill se vad du skriver vid indata i REPL kan du använda `jline`[9]

[9] github.com/jline/jline2

och klassen `jline.console.ConsoleReader`[10]. Då får du dessutom editeringsfunktioner vid inmatning med t.ex. Ctrl+A och Ctrl+K så som i en vanlig unixterminal. Med pil upp och pil ner kan du bläddra i inmatningshistoriken.

```
scala> val scan = new java.util.Scanner(System.in)
scala> scan.next
scala> scan.nextInt
scala> val cr = new jline.console.ConsoleReader
scala> cr.readLine
scala> cr.readLine("> ")
scala> cr.readLine("Ange tal: ").toInt
scala> scala.util.Try{cr.readLine("Ange tal: ").toInt}.toOption
```

a) Prova ovan rader i REPL. Vad händer om du matar in bokstäver i stället för siffror på sista raden ovan? (Mer om `Option` i kapitel **??**).

b) Skriv ett funktion `readPalindromLoop` som låter användaren mata in strängar och som kollar om de är palindromer så som nedan REPL-körning indikerar. Skriv funktionen i en editor och klistra in den i REPL enligt nedan istället för `???`

```
scala> val cr = new jline.console.ConsoleReader
scala> def isPalindrome(s: String): Boolean = s == s.reverse
scala> :paste
// Entering paste mode (ctrl-D to finish)

def readPalindromLoop: Unit = ???

// Exiting paste mode, now interpreting.

readPalindromLoop: Unit

scala> readPalindromLoop
Ange sträng följt av <Enter>
Programmet avslutas med tom sträng + <Enter>
> gurka
gurka är ingen palindrom
> dallassallad
dallassallad är en palindrom!
>
Tack och hej!
scala>
```

c) Skapa ett objekt med inläsningsstöd enligt nedan specifikation. Objektet ska delegera implementationerna till ett attribut **`private val`** `reader` som innehåller en referens till ett `ConsoleReader`-objekt.

Specification **`termutil`**

```
object termutil {
  /** Reads one line from terminal input. */
  def readLine: String = ???
```

[10] jline.github.io/jline2/apidocs/reference/jline/console/ConsoleReader.html

```
  /** Prints prompt and reads one line. */
  def readLine(prompt: String): String = ???

  /** Reads one line and converts it to an Int.
   *  If a non-integer is input, a NumberFormatException is thrown.  */
  def readInt: Int = ???

  /** Prints prompt, reads one line and converts it to an Int.
   *  If a non-integer is input, a NumberFormatException is thrown.  */
  def readInt(prompt: String): Int = ???

  /** Reads one line and converts it to an Option[Int]
   *  with Some integer or None if the input cannot be converted.  */
  def readIntOpt: Option[Int] = ???

  /** Prints prompt, reads one line and converts it to an Option[Int]
   *  with Some integer or None if the input cannot be converted.  */
  def readIntOpt(prompt: String): Option[Int] = ???
}
```

Biblioteket jline finns inbyggd i REPL men om du vill kompilera din kod separat kan du ladda ner jar-filen här: repo1.maven.org/maven2/jline/jline/2.10/ eller så hittar du den bland dina Scala-installationsfiler och kan kopiera filen till dit du vill ha den. Placera jline-jar-filen i samma bibliotek som din kod, eller lägg den i ett biblioteket där du vill ha den och placera jarfilen på classpath med optionen `-cp` när du kompilerar ungefär så här:

```
scalac -cp "lib/jline-2.10.jar" termutil.scala
```

6. Övning: `classes`

Mål

- ☐ Kunna deklarera klasser med klassparametrar.
- ☐ Kunna skapa objekt med **`new`** och konstruktorargument.
- ☐ Förstå innebörden av referensvariabler och värdet **`null`**.
- ☐ Förstå innebörden av begreppen instans och referenslikhet.
- ☐ Kunna använda nyckelordet **`private`** för att styra synlighet i primärkonstruktor.
- ☐ Förstå i vilka sammanhang man kan ha nytta av en privat konstruktor.
- ☐ Kunna implementera en klass utifrån en specikation.
- ☐ Förstå skillnaden mellan referenslikhet och strukturlikhet.
- ☐ Känna till hur case-klasser hanterar likhet.
- ☐ Förstå nyttan med att möjliggöra framtida förändring av attributrepresentation.
- ☐ Känna till begreppen getters och setters.
- ☐ Känna till accessregler för kompanjonsobjekt.
- ☐ Känna till skillnaden mellan == och `eq`, samt `!=` versus `ne`.

Förberedelser

- ☐ Studera begreppen i kapitel **??**.

6.1 Grunduppgifter

Uppgift 1. *Instansiering med **`new`** och värdet **`null`**.* Man skapar instanser av klasser med **`new`**. Då anropas konstruktorn och plats reserveras i datorns minne för objektet. Variabler av referenstyp som inte refererar till något objekt har värdet **`null`**.

a) Vad händer nedan? Vilka rader ger felmeddelande och i så fall hur lyder felmeddelandet?

```
1 scala> class Gurka(val vikt: Int)
2 scala> var g: Gurka = null
3 scala> g.vikt
4 scala> g = new Gurka(42)
5 scala> g.vikt
6 scala> g = null
7 scala> g.vikt
```

b) Rita minnessituationen efter raderna 2, 4, 6.

Uppgift 2. *Klasser och instanser.*

a) Vad händer nedan?

```
1 scala> :pa
2 class Arm(val ärTillVänster: Boolean)
3 class Ben(val ärTillVänster: Boolean)
```

```
4  class Huvud(val harHår: Boolean)
5  class Rymdvarelse {
6    var arm1 = new Arm(true)
7    var arm2 = new Arm(false)
8    var ben1 = new Ben(true)
9    var ben2 = new Ben(false)
10   var huvud1 = new Huvud(false)
11   var huvud2 = new Huvud(true)
12   def ärSkallig = !huvud1.harHår && !huvud2.harHår
13 }
14 scala> val alien = new RymdVarelse
15 scala> alien.ärSkallig
16 scala> val predator = new RymdVarelse
17 scala> predator.ärSkallig
18 scala> predator.huvud2 = alien.huvud1
19 scala> predator.ärSkallig
```

b) Rita minnessituationen efter rad 18.

c) Vad händer så småningom med det ursprungliga huvud2-objektet i predator efter tilldelningen på rad 18? Går det att referera till detta objekt på något sätt?

Uppgift 3. *Synlighet i primärkonstruktorer.* Undersök nedan vad nyckelorden **`val`** och **`private`** får för konsekvenser. Förklara vad som händer. Vilka rader ger vilka felmeddelanden?

```
1  scala> class Gurka1(vikt: Int)
2  scala> new Gurka1(42).vikt
3  scala> class Gurka2(val vikt: Int)
4  scala> new Gurka2(42).vikt
5  scala> class Gurka3(private val vikt: Int)
6  scala> new Gurka3(42).vikt
7  scala> class Gurka4(private val vikt: Int, kompis: Gurka4){
8           def kompisVikt = kompis.vikt
9         }
10 scala> val ingenGurka: Gurka4 = null
11 scala> new Gurka4(42, ingenGurka).kompisVikt
12 scala> new Gurka4(42, new Gurka4(84, null)).kompisVikt
13 scala> class Gurka5(private[this] val vikt: Int, kompis: Gurka5){
14          def kompisVikt = kompis.vikt
15        }
16 scala> class Gurka6 private (vikt: Int)
17 scala> new Gurka6(42)
18 scala> :pa
19 class Gurka7 private (var vikt: Int)
20 object Gurka7 {
21   def apply(vikt: Int) = {
22     require(vikt >= 0, s"negativ vikt: $vikt")
23     new Gurka7(vikt)
24   }
25 }
26 scala> new Gurka7(-42)
27 scala> Gurka7(-42)
28 scala> val g = Gurka7(42)
```

```
scala> g.vikt
scala> g.vikt = -1
scala> g.vikt
```

Uppgift 4. *Egendefinierad setter kombinerat med privat konstruktor.*

a) Förklara vad som händer nedan. Vilka rader ger vilka felmeddelanden?

```
scala> :pa
class Gurka8 private (private var _vikt: Int) {
  def vikt = _vikt
  def vikt_=(v: Int): Unit = {
    require(v >= 0, s"negativ vikt: $v")
    _vikt = v
  }
}

object Gurka8 {
  def apply(vikt: Int) = {
    require(vikt >= 0, s"negativ vikt: $vikt")
    new Gurka8(vikt)
  }
}
scala> val g = Gurka8(-42)
scala> val g = Gurka8(42)
scala> g.vikt
scala> g.vikt = 0
scala> g.vikt = -1
scala> g.vikt += 42
scala> g.vikt -= 1000
```

b) Vad är fördelen med möjligheten att skapa egendefienerade setters?

Uppgift 5. *En oföränderlig kvadrat med alternativ fabriksmetod.*

a) Implementera klassen `Square` enligt nedan specifikation. Gör implementationen i en kodeditor, så som `gedit`, och klistra in klassen i Scala REPL efter kommandot `:pa` (förkortning av `:paste`). På så sätt blir **`object`** `Square` ett kompanjonsobjekt till **`class`** `Square`.

Specification **`Square`**

```
/** A class representing a square object with position and side. */
class Square(val x: Int, val y: Int, val side: Int) {
  /** The area of this Square */
  val area: Int = ???

  /** Creates a new Square moved to position (x + dx, y + dy) */
  def move(dx: Int, dy: Int): Square = ???

  /** Tests if this Square has equal size as that Square */
  def isEqualSizeAs(that: Square): Boolean = ???

  /** Multiplies the side with factor and rounded to nearest integer */
  def scale(factor: Double): Square = ???
```

```
  /** A string representation of this Square */
  override def toString: String = ???
}

object Square {
  /** A square placed in origin with size 1 */
  val unit: Square = ???

  /** Constructs a new Square object at (x, y) with size side */
  def apply(x: Int, y: Int, side: Int): Square = ???

  /** Constructs a new Square object at (0, 0) with side 1 */
  def apply(): Square = ???
}
```

b) Testa din kvadrat enligt nedan. Förklara vad som händer.

```
scala> val (s1, s2) = (Square(), Square(1, 10, 1))
scala> val s3 = s1.move(1,-5)
scala> s1 isEqualSizeAs s3
scala> s2 isEqualSizeAs s1
scala> s1 isEqualSizeAs Square.unit
scala> s2.scale(math.Pi) isEqualSizeAs s2
scala> s2.scale(math.Pi) isEqualSizeAs s2.scale(math.Pi)
```

Uppgift 6. *Referenslikhet versus strukturlikhet.* Metoden == på case-klasser ger **strukturlikhet** (även kallad innehållslikhet) så att *innehållet* i klassens klassparametrar jämförs om de har lika värde, medan för vanliga klasser ger metoden == **referenslikhet** där olika objekt är olika även om de har samma innehåll (om man inte överskuggar metoden `equals` som anropas av == vilket vi ska titta närmare på i kapitel **??**).

```
scala> class GurkaRef(val vikt: Int)
scala> case class GurkaStrukt(val vikt: Int)
scala> val a = new GurkaRef(42)
scala> val b = new GurkaRef(42)
scala> val c = new GurkaStrukt(42)
scala> val d = new GurkaStrukt(42)
scala> a == b
scala> c == d
```

a) Förklara vad som händer ovan.

b) Istället för ==, prova metoden `eq` på objekten ovan. Metoden `eq` ger alltid referenslikhet (även om byter ut metoden `equals`).

Uppgift 7. *Klassen `Point` med case-klass.*

a) Implementera klassen `Point` som en oföränderlig case-klass med heltalsattributen `x` och `y`.

b) Lägg till metoden `distanceTo(that: Point): Double` som räknar ut avståndet till en annan punkt med hjälp av `math.hypot`.

c) Lägg till metoden `distanceTo(x: Int, y: Int): Double` som räknar ut avståndet till koordinaterna x och y med hjälpa av metoden i föregående deluppgift.

d) Lägg till metoden `move(dx: Int, dy: Int): Point` som skapar en ny punkt på translaterad position enligt delta-koordinaterna `dx` och dy.

e) Lägg till ett kompanjonsobjekt med medlemmen `val origin` som ger en punkt i origo.

f) Undersök metoderna ==, !=, eq och ne och förklara vad som händer nedan:

```
scala> Point(1, 2) == Point(1, 3)
scala> Point(1, 2) != Point(1, 3)
scala> Point(1, 2) == Point(1, 2)
scala> Point(1, 2) != Point(1, 2)
scala> Point.origin.move(1, 1) == Point.origin.move(1, 1)
scala> Point.origin.move(1, 1).move(1, 1) != Point(2, 2)
scala> Point(0, 0) eq Point(0, 0)
scala> Point(0, 0) ne Point(0, 0)
scala> Point.origin eq Point.origin
scala> Point.origin ne Point.origin
scala> val p1 = Point(0, 0)
scala> val p2 = p1
scala> p1 eq p2
```

g) Vad ger `Point.origin eq Point.origin` för resultat om `origin` istället implementeras som `def origin: Point = Point(0, 0)`

h) Vad är det för skillnad på strukturlikhet och referenslikhet?

Uppgift 8. Ändra representationen av positionen i klassen `Square` från deluppgift 5 till att vara en `Point` från deluppgift 7.

Uppgift 9. *Case-klassen `Point` med 2-tupel.* I ett utvecklingsprojekt vill man ändra representationen av positionen i den gamla klassen `case class Point(x: Int, y: Int)` så att positionen istället i den uppdaterade klassen representeras av en 2-tupel. Man kan då vid konstruktion utnyttja att n-tupler som parameter även kan skrivas som en parameterlista med n argument, varför både `Point(1,2)` och `Point((1,2))` fungerar fint. Samtidigt vill man att befintlig kod som fortfarande använder `x` och `y` ska fungera utan ändringar. Implementera den nya `Point` enligt specifikationen nedan.

Specification **`Point`**

```
/** A 2-dimensional immutable position p in an integer coordinate system */
case class Point(p:(Int, Int)) {
  /** The x-axis position of this Point */
  val x: Int = ???

  /** The y-axis position of this Point */
  val y: Int = ???

  /** The distance to another Point that */
```

```
  def distanceTo(that: Point): Double = ???

  /** The distance to another 2-tuple that representing (x, y). */
  def distanceTo(that: (Int, Int)): Double = ???

  /** A new Point that is moved (dx, dy) */
  def move(dxdy: (Int, Int)): Point = ???
}

object Point {
  /** A Point object at position (0, 0) */
  val origin: Point = ???
}
```

✎ **Uppgift 10.** Vad behöver du ändra i klassen `Square` från uppgift 8 för att den ska fungera med en `Point` med 2-tupel från uppgift 9?

Uppgift 11. *Objekt med föränderligt tillstång (eng. mutable state).* Du ska implementera en modell av en hoppande groda som uppfyller följande krav:

1. Varje grodobjekt ska hålla reda på var den är.
2. Varje grodobjekt ska hålla reda på hur långt grodan hoppat totalt.
3. Varje grodobjekt ska kunna beräkna hur långt det är mellan grodans nuvarande position och utgångsläget.
4. Alla grodor börjar sitt hoppande i origo.
5. En groda kan hoppa enligt två metoder:
   - relativ förflyttning enligt parametrarna `dx` och `dy`,
   - slumpmässig förflyttning $[1, 10]$ i x-led och $[1, 10]$ i y-led.

a) Implementera klassen `Frog` enligt nedan specifikation och ovan krav.
Tips:

- Om namnet man vill ge ett privat föränderligt attribut ”krockar” med ett metodnamn, är det vanligt att man börjar attributets namn med understreck, t.ex. `private var _x` för att på så sätt undkomma namnkonflikten.
- Inför en metod i taget och klistra in den nya grodan i REPL efter varje utvidgning och testa.

Specification **`Frog`**

```
class Frog private (initX: Int = 0, initY: Int = 0) {
  def jump(dx: Int, dy: Int): Unit = ???
  def x: Int = ???
  def y: Int = ???
  def randomJump: Unit = ???
  def distanceToStart: Double = ???
  def distanceJumped: Double = ???
  def distanceTo(that: Frog): Double = ???
}
object Frog {
  def spawn(): Frog = ???
}
```

b) Skriv ett testhuvudprogram som kontrollerar så att alla krav är uppfyllda och att alla metoder fungerar som de ska.

c) Vad kallas en metod som enbart returnerar värdet av ett privat attribut? 

d) Hur kan man från en metods signatur få en ledtråd om att ett objekt har föränderligt tillstånd (eng. *mutable state*)? 

e) Inför setters för attributen som håller reda på x- och y-postitionen. Förändringar av positionen i x- eller y-led ska räknas som ett hopp och alltså registreras i det attribut som håller reda på det ackumulerade hoppavståndet.

f) Simulera ett massivt grodhoppande med krockdetektering genom att skapa 100 grodor som till att börja med är placerade på x-axeln med avståndet 8 längdenheter mellan sig. Låt grodorna i en `while`-sats hoppa slumpmässigt tills någon groda befinner sig närmare än 0.5 längdenheter som är definitionen på att de har krockat. Räkna hur många looprundor som behövs innan något grodpar krockar och skriv ut antalet.
Tips: Börja med pseudokod på papper. Använd en grodvektor.

6.2 Extrauppgifter

Uppgift 12. *En kvadratklass med föränderligt tillstånd (eng. mutable state).* Webbshoppen UberSquare säljer flyttbara kvadrater. I affärsmodellen ingår att ta betalt per förflytttning. Du ska hjälpa UberSquare med att utveckla en enkel systemprototyp.

a) Implementera `Square` enligt nedan specifikation, under uppfyllandet av följande krav:

1. Till skillnad från uppgift 5 ska du nu göra en kvadrat med föränderligt tillstånd (eng. *mutable state*). I stället för att vid förflyttning returnera ett nytt kvadratobjekt, returneras `Unit` i samband med att privata attribut uppdateras.
2. Du ska införa funktionalitet som räknar antalet förflyttningar som gjorts för varje kvadrat som skapats och även räkna ut det totala antalet förflyttningar som någonsin gjorts.
3. Varje gång förflyttning sker adderas en kostnad till den ackumulerade kostnaden för respektive kvadrat. Kostnaden för varje förflyttning är avståndet till ursprungsläget multiplicerat med storleken på kvadraten.

Specification **Square**

```
/** A mutable and expensive Square. */
class Square private (val initX: Int, val initY: Int, val initSide: Int) {

  private var nMoves = 0;
  private var sumCost = 0.0;
  private var _x = initX;
  private var _y = initY;
  private var _side = initSide;

  private def addCost: Unit = {
   sumCost += ???
  }

  /** The current position on the x axis */
  def x: Int = ???

  /** The current position on the y axis */
  def y: Int = ???

  /** The size of this Square */
  def side = ???

  /** Scales the side of this square and rounds it to nearest integer */
  def scale(factor: Double): Unit = ???

  /** Moves this square to position (x + xd, y + dy) */
  def move(dx: Int, dy: Int): Unit = ???

  /** Moves this square to position (x, y) */
  def moveTo(x: Int, y: Int): Unit = ???
```

```
  /** The accumulated cost of this Square */
  def cost: Double = ???

  /** Reset the cost of this Square */
  def pay: Unit = ???

  /** A string representation of this Square */
  override def toString: String =
    s"Square[($x, $y), side: $side, #moves: $nMoves times, cost: $sumCost]"
}

object Square {
  private var created = Vector[Square]()

  /** Constructs a new Square object at (x, y) with size side */
  def apply(x: Int, y: Int, side: Int): Square = {
    require(side >= 0, s"side must be positive: $side")
    ???
  }

  /** Constructs a new Square object at (0, 0) with side 1 */
  def apply(): Square = apply(0, 0, 1)

  /** The total number of moves that have been made for all squares. */
  def totalNumberOfMoves: Int = ???

  /** The total cost of all squares. */
  def totalCost: Double = ???
}
```

b) Testa din kvadratprototyp i REPL enligt nedan:

```
scala> val xs = Vector.fill(10)(Square())
scala> xs.foreach(_.move(2,3))
scala> xs.foreach(_.scale(2.9))
scala> val (m, c) = (Square.totalNumberOfMoves, Square.totalCost)
m: Int = 10
c: Double = 36.055512754639885
```

6.3 Fördjupningsuppgifter

Uppgift 13. *Hjälpkonstruktor.* I uppgift 5 erbjöds ett alternativt sätt att skapa `Square` med en extra fabriksmetod med namnet `apply` i kompanjonsobjektet. Ett annat sätt att göras detta på, som i Scala är mindre vanligt (men i Java är desto vanligare), är att definiera flera konstruktorer innuti klassen. I Scala kallas en sådan extra konstruktor för **hjälpkonstruktor** (eng. *auxiliary constructor*).

En hjälpkonstruktor skapar man i Scala genom att definiera en metod som har det speciella namnet `this`, alltså en deklaration `def this(...) = ...` Hjälponstruktorer måste börja med att anropa en annan konstruktor, antingen den primära konstruktorn eller en tidigare definierad hjälpkonstruktor.

a) Läs mer om hjälpkonstruktorer här:
www.artima.com/pins1ed/functional-objects.html#6.7

b) Hitta på en egen uppgift med hjälpkonstruktorer, baserat på någon av klasserna i tidigare övningar.

7. Övning: `traits`

Mål

- ☐ Förstå följande begrepp:
  - ☐ bastyp,
  - ☐ supertyp,
  - ☐ subtyp,
  - ☐ abstrakt typ,
  - ☐ polymorfism.
- ☐ Kunna deklarera och använda en arvshierarki i flera nivåer med nyckelordet **`extends`**.
- ☐ Kunna deklarera och använda inmixning med flera traits och nyckelordet **`with`**.
- ☐ Kunna deklarera och känna till nyttan med finala klasser och finala attribut och nyckelordet **`final`**.
- ☐ Känna till synlighetsregler vid arv och nyttan med privata och skyddade attribut.
- ☐ Kunna deklarera och använda skyddade attribute med nyckelordet **`protected`**.
- ☐ Känna till hur typtester och typkonvertering vid arv kan göras med metoderna `isInstanceOf` och `asInstanceOf` och känna till att detta görs bättre med **`match`** (som kommer i kapitel **??**).
- ☐ Känna till begreppet anonym klass.
- ☐ Kunna deklarera och använda överskuggade metoder med nyckelordet **`override`**.
- ☐ Känna till reglerna som gäller vid överskuggning av olika sorters medlemmar.
- ☐ Kunna deklarera och använda hierarkier av klasser där konstruktorparametrar överförs till superklasser.
- ☐ Kunna deklarera och använda uppräknade värden med case-objekt och gemensam bastyp.

Förberedelser

- ☐ Studera begreppen i kapitel **??**.

7.1 Grunduppgifter

Uppgift 1. *Gemensam bastyp.* Man vill ofta lägga in objekt av olika typ i samma samling.

```
1 class Gurka(val vikt: Int)
2 class Tomat(val vikt: Int)
3 val gurkor = Vector(new Gurka(100), new Gurka(200))
4 val grönsaker = Vector(new Gurka(300), new Tomat(42))
```

a) Om en samling innehåller objekt av flera olika typer försöker kompilatorn härleda den mest specifika typen som objekten har gemensamt. Vad blir det för typ på värdet `grönsaker` ovan?

b) Försök ta reda på summan av vikterna enligt nedan. Vad ger andra raden för felmeddelande? Varför?

```
gurkor.map(_.vikt).sum
grönsaker.map(_.vikt).sum
```

c) Vi kan göra så att vi kan komma åt vikten på alla grönsaker genom att ge gurkor och tomater en gemensam bastyp som de olika konkreta grönsakstyperna utvidgar med nyckelordet **`extends`**. Man säger att subtyperna `Gurka` och `Tomat` **ärver** egenskaperna hos supertypen `Grönsak`.

Attributet `vikt` i traiten `Grönsak` nedan initialiseras inte förrän konstruktorerna anropas när vi gör **`new`** på någon av klasserna `Gurka` eller `Tomat`.

```
trait Grönsak { val vikt: Int }
class Gurka(val vikt: Int) extends Grönsak
class Tomat(val vikt: Int) extends Grönsak
val gurkor = Vector(new Gurka(100), new Gurka(200))
val grönsaker = Vector(new Gurka(300), new Tomat(42))
```

d) Vad blir det nu för typ på variabeln `grönsaker` ovan?

e) Fungerar det nu att räkna ut summan av vikterna i `grönsaker` med `grönsaker.map(_.vikt).sum`?

f) En trait liknar en klass, men man kan inte instansiera den och den kan inte ha några parametrar. En typ som inte kan instansieras kallas **abstrakt** (eng. *abstract*). Vad blir det för felmeddelande om du försöker göra **`new`** på en trait enligt nedan?

```
trait Grönsak{ val vikt: Int }
new Grönsak
```

g) Traiten `Grönsak` har en abstrakt medlem `vikt`. Den sägs vara abstrakt eftersom den saknar definition – medlemmen har bara ett namn och en typ men inget värde. Du kan instansiera den abstrakta traiten `Grönsak` om du fyller i det som ”fattas”, nämligen ett värde på `vikt`. Man kan fylla på det som fattas i genom att ”hänga på” ett block efter typens namn vid instansiering. Man får då vad som kallas en **anonym** klass, i detta fall en ganska konstig grönsak som inte är någon speciell sorts grönsak med som ändå har en vikt.

Vad får `anonymGrönsak` nedan för typ och strängrepresenation?

```
val anonymGrönsak = new Grönsak { val vikt = 42 }
```

Uppgift 2. *Polymorfism i samband med arv.* Polymorfism betyder ”många skepnader”. I samband med arv innebär det att flera subtyper, till exempel `Ko` och `Gris`, kan hanteras gemensamt som om de vore instanser av samma supertyp, så som `Djur`. Subklasser kan implementera en metod med samma namn på

olika sätt. Vilken metod som exekveras bestäms vid körtid beroende på vilken subtyp som instansieras. På så sätt kan djur komma i många skepnader.

a) Implementera funktionen `skapaDjur` nedan så att den returnerar antingen en ny Ko eller en ny Gris med lika sannolikhet.

```
scala> trait Djur { def väsnas: Unit }
scala> class Ko   extends Djur { def väsnas = println("Muuuuuuu") }
scala> class Gris extends Djur { def väsnas = println("Nöffnöff") }
scala> def skapaDjur: Djur = ???
scala> val bondgård = Vector.fill(42)(skapaDjur)
scala> bondgård.foreach(_.väsnas)
```

b) Lägg till ett djur av typen Häst som väsnas på lämpligt sätt och modifiera `skapaDjur` så att det skapas kor, grisar och hästar med lika sannolikhet.

Uppgift 3. *Bastypen `Shape` och subtyperna `Rectangle` och `Circle`.* Du ska nu skapa ett litet bibliotek för geometriska former med oföränderliga objekt implementerade med hjälp av case-klasser. De geometriska formerna har en gemensam bastyp kallad `Shape`. Skriv nedan kod i en editor och klistra sedan in den i REPL med kommandot `:paste`.

```
case class Point(x: Double, y: Double) {
  def move(dx: Double, dy: Double): Point = Point(x + dx, y + dy)
}

trait Shape {
  def pos: Point
  def move(dx: Double, dy: Double): Shape
}

case class Rectangle(
  pos: Point,
  dx: Double,
  dy: Double
) extends Shape {
  override def move(dx: Double, dy: Double): Rectangle =
    Rectangle(pos.move(dx, dy), this.dx, this.dy)
}

case class Circle(pos: Point, radius: Double) extends Shape {
  override def move(dx: Double, dy: Double): Circle =
    Circle(pos.move(dx, dy), radius)
}
```

a) Instansiera några cirklar och rektanglar och gör några relativa förflyttningar av dina instanser genom att anropa `move`.

b) Lägg till metoden `moveTo` i `Point`, `Shape`, `Rectangle` och `Circle` som gör en absolut förflyttning till koordinaterna `x` och `y`. Klistra in i REPL och testa

på några instanser av `Rectangle` och `Circle`.

c) Lägg till metoden `distanceTo(that: Point): Double` i case-klassen `Point` som räknar ut avståndet till en annan punkt med hjälp av `math.hypot`. Klistra in i REPL och testa på några instanser av `Point`.

d) Lägg till en konkret metod `distanceTo(that: Shape): Double` i traiten `Shape` som räknar ut avståndet till positionen för en annan Shape. Klistra in i REPL och testa på några instanser av `Rectangle` och `Circle`.

Uppgift 4. *Inmixning.* Man kan utvidga en klass med multipla traits med nyckelordet **`with`**. På så sätt kan man fördela medlemmar i olika traits och återanvända gemensamma koddelar genom så kallad **inmixning**, så som nedan exempel visar.

En alternativ fågeltaxonomi, speciellt populär i Skåne, delar in alla fåglar i två specifika kategorier: Kråga respektive Ånka. Krågor kan flyga men inte simma, medan Ånkor kan simma och oftast även flyga. Fågel i generell, kollektiv bemärkelse kallas på gammal skånska för Fyle.[11] Skriv in nedan kod i en editor och spara den för kommande uppgifter. Klistra in koden i REPL med kommandot `:paste`.

```
trait Fyle {
  val läte: String
  def väsnas: Unit = print(läte * 2)
  val ärSimkunnig: Boolean
  val ärFlygkunnig: Boolean
}

trait KanSimma       { val ärSimkunnig = true }
trait KanInteSimma   { val ärSimkunnig = false }
trait KanFlyga       { val ärFlygkunnig = true }
trait KanKanskeFlyga { val ärFlygkunnig = math.random < 0.8 }

class Kråga extends Fyle with KanFlyga with KanInteSimma {
  val läte = "krax"
}

class Ånka extends Fyle with KanSimma with KanKanskeFlyga {
  val läte = "kvack"
  override def väsnas = print(läte * 4)
}
```

a) En flitig ornitolog hittar 42 fåglar i en perfekt skog där alla fågelsorter är lika sannolika, representerat av vektorn `fyle` nedan. Skriv i REPL ett uttryck som undersöker hur många av dessa som är flygkunniga Ånkor, genom att använda metoderna `filter`, `isInstanceOf`, `ärFlygkunnig` och `size`.

[11] www.klangfix.se/ordlista.htm

```
scala> val fyle =
         Vector.fill(42)(if (math.random > 0.5) new Kråga else new Ånka)
scala> fyle.foreach(_.väsnas)
scala> val antalFlygånkor: Int = ???
```

b) Om alla de fåglar som ornitologen hittade skulle väsnas exakt en gång var, hur många krax och hur många kvack skulle då höras? Använd metoderna `filter` och `size`, samt predikatet `ärSimKunnig` för att beräkna antalet krax respektive kvack.

```
scala> val antalKrax: Int = ???
scala> val antalKvack: Int = ???
```

Uppgift 5. *Finala klasser.* Om man vill förhindra att man kan göra **`extends`** på en klass kan man göra den final genom att placera nyckelordet **`final`** före nyckelordet **`class`**.

a) Eftersom klassificeringen av fåglar i uppgiften ovan i antingen Ånkor eller Krågor är fullständig och det inte finns några subtyper till varken Ånkor eller Krågor är det lämpligt att göra dessa finala. Ändra detta i din kod.

b) Testa att ändå försöka göra en subklass `Simkråga` **`extends`** `Kråga`. Vad ger kompilatorn för felmeddelande om man försöker utvidga en final klass?

Uppgift 6. *Accessregler vid arv och nyckelordet* ***`protected`****.* Om en medlem i en supertyp är privat så kan man inte komma åt den i en subklass. Ibland vill man att subklassen ska kunna komma åt en medlem även om den ska vara otillgänglig i annan kod.

```
trait Super {
  private val minHemlis = 42
  protected val vårHemlis = 42
}
class Sub extends Super {
  def avslöja = minHemlis
  def kryptisk = vårHemlis * math.Pi
}
```

a) Vad blir felmeddelandet när klassen `Sub` försöker komma åt `minHemlis`?

b) Deklarera `Sub` på nytt, men nu utan den förbjudna metoden `avslöja`. Vad blir felmeddelandet om du försöker komma åt `vårHemlis` via en instans av klassen `Sub`? Prova till exempel med `(new Sub).vårHemlis`

c) Fungerar det att anropa metoden `kryptisk` på instanser av klassen `Sub`?

Uppgift 7. *Användning av* ***`protected`****.* Den flitige ornitologen från uppgift 4 ska ringmärka alla 42 fåglar hen hittat i skogen. När hen ändå håller på bestämmer hen att i även försöka ta reda på hur mycket oväsen som skapas av respektive fågelsort. För detta ändamål apterar den flitige ornitologen en linuxdator på allt infångat fyle. Du ska hjälpa ornitologen att skriva programmet.

a) Inför en `protected var räknaLäte` i traiten `Fyle` och skriv kod på lämpliga ställen för att räkna hur många läten som respektive fågelinstans yttrar.

b) Inför en metod `antalLäten` som returnerar antalet krax respektive kvack som en viss fågel yttrat sedan dess skapelse.

c) Varför inte använda `private` i stället for `protected`?

d) Varför är det bra att göra räknar-variabeln oåtkomlig från ”utsidan”?

Uppgift 8. *Typtester med `isInstanceOf` och typkonvertering med `asInstanceOf`.* Gör nedan deklarationer.

```
scala> trait A; trait B extends A; class C extends B; class D extends B
scala> val (c, d) = (new C, new D)
scala> val a: A = c
scala> val b: B = d
```

a) Rita en bild över vilka typer som ärver vilka.

b) Vilket resultat ger dessa typtester? Varför?

```
scala> c.isInstanceOf[C]
scala> c.isInstanceOf[D]
scala> d.isInstanceOf[B]
scala> c.isInstanceOf[A]
scala> b.isInstanceOf[A]
scala> b.isInstanceOf[D]
scala> a.isInstanceOf[B]
scala> c.isInstanceOf[AnyRef]
scala> c.isInstanceOf[Any]
scala> c.isInstanceOf[AnyVal]
scala> c.isInstanceOf[Object]
scala> 42.isInstanceOf[Object]
scala> 42.isInstanceOf[Any]
```

c) Vilka av dessa typkonverteringar ger felmeddelande? Vilket och varför?

```
scala> c.asInstanceOf[B]
scala> c.asInstanceOf[A]
scala> d.asInstanceOf[C]
scala> a.asInstanceOf[B]
scala> a.asInstanceOf[C]
scala> a.asInstanceOf[D]
scala> a.asInstanceOf[E]
scala> b.asInstanceOf[A]
```

Uppgift 9. *Regler för `override`, `private` och `final`.*

a) Undersök överskuggningning av abstrakta, konkreta, privata och finala medlemmar genom att skriva in raderna nedan en i taget i REPL. Vilka rader ger felmeddelande? Varför? Vid felmeddelande: notera hur felmeddelandet lyder och ändra deklarationen av den felande medlemmen så att koden blir kompilerbar (eller om det är enda rimliga lösningen: ta bort den felaktiga medlemmen), innan du provar efterkommande rad.

```
trait Super1 { def a: Int; def b = 42; private def c = "hemlis" }
class Sub2 extends Super1 { def a = 43; def b = 43; def c = 43 }
class Sub3 extends Super1 { def a = 43; override def b = 43 }
class Sub4 extends Super1 { def a = 43; override def c = "43" }
trait Super5 { final def a: Int; final def b = 42 }
class Sub6 extends Super5 { override def a = 43; def b = 43 }
class Sub7 extends Super5 { def a = 43; override def b = 43 }
class Sub8 extends Super5 { def a = 43; override def c = "43" }
trait Super9 { val a: Int; val b = 42; lazy val c: String = "lazy" }
class Sub10 extends Super9 { override def a = 43; override val b = 43 }
class Sub11 extends Super9 { val a = 43; override lazy val b = 43 }
class Sub12 extends Super9 { val a = 43; override var b = 43 }
class Sub13 extends Super9 { val a = 43; override lazy val c = "still lazy" }
class SubSub extends Sub13 { override val a = 44}
trait Super14 { var a: Int; var b = 42; var c: String }
class Sub15 extends Super14 { def a = 43; override var b = 43; val c = "?" }
```

b) Skapa instanser av klasserna Sub3, Sub13 och SubSub från ovan deluppgift och undersök alla medlemmarnas värden för respektive instans. Förklara varför de har dessa värden.

c) Läs igenom reglerna i kapitel **??** om vad som gäller vid arv och överskuggning av medlemmar. Gör några egna undersökningar i REPL som försöker bryta mot någon regel som inte testades i deluppgift a.

Uppgift 10. *Supertyp med parameter.* En trait kan inte ha någon parameter. Vill man ha en parameter till supertypen måste man använda en klass istället, enligt nedan exempel.

Utbildningsdepartementet vill i sitt system hålla koll på vissa personer och skapar därför en klasshierarki enligt nedan. Skriv in koden i en editor och klipp sedan in den i REPL.

```
class Person(val namn: String)

class Akademiker(
  namn: String,
  val universitet: String) extends Person(namn)

class Student(
  namn: String,
  universitet: String,
  program: String) extends Akademiker(namn, universitet)

class Forskare(
  namn: String,
  universitet: String,
  titel: String) extends Akademiker(namn, universitet)
```

a) Deklarera fyra olika `val`-variabler med lämpliga namn som refererar till olika instanser av alla olika klasser ovan och ge attributen valfria initialvärden

genom olika parametrar till konstruktorerna.

b) Skriv två satser: en som först stoppar in instanserna i en `Vector` och en som sedan loopar igenom vektorn och skriv ut alla instansers `toString` och `namn`.

c) Utbildningsdepartementet vill att det inte ska gå att instansiera objekt av typerna `Person` och `Akademiker`. Det kan åstadkommas genom att placera nyckelordet **`abstract`** före **`class`**. Uppdatera koden i enlighet med detta. Vilket blir felmeddelande om man försöker instansiera en **`abstract class`**?

d) Utbildningsdeparetementet vill slippa implementera `toString` och slippa skriva **`new`** vid instansiering. Gör därför om typerna `Student` och `Forskare` till case-klasser. *Tips:* För att undkomma ett kompileringsfel (vilket?) behöver du använda **`override val`** på lämpligt ställe.

Skapa instanser av de nya case-klasserna `Student` och `Forskare` och skriv ut deras `toString`. Hur ser utskriften ut?

e) Eftersom `Person` och `Akademiker` nu är abstrakta, vill utbildningsdepartementet att du gör om dessa typer till traits med abstrakta attribut istället för klasser. Du kan då undvika **`override val`** i klassparametrarna till de konkreta case-klasserna.

Man inför också en case-klass `IckeAkademiker` som man tänker använda i olika statistiska medborgarundersökningar.

Dessutom förser man alla personer med ett personnummer representerat som en `Int`.

Hur ser utbildningsdepartementets kod ut nu, efter alla ändringar? Skriv ett testprogram som skapar några instanser och skriver ut deras attribut.

f) I vilka sammanhang är det nödvändigt att använda en **`trait`** respektive en **`class`**.

Uppgift 11. *Uppräknade värden.* Ett sätt att hålla reda på uppräknade värden, t.ex. färgen på olika kort i en kortlek, är att använda olika heltal som får representera de olika värdena, till exempel så här:[12]

```
object Färg {
  val Spader = 1
  val Hjärter = 2
  val Ruter = 3
  val Klöver = 4
}
```

Dessa kan sedan användas som parametrar till denna case-klass vid skapande av kortobjekt:

```
case class Kort(färg: Int, valör: Int)
```

[12] Om namnkonventioner för konstanter i Scala: läs under rubriken ”Constants, Values, Variable and Methods” här docs.scala-lang.org/style/naming-conventions.html

Man kan hålla reda på färgen med en variabel av typen `Int` och tilldela den en viss färg med ovan konstanter. Och när man skapar ett kort behöver man inte komma ihåg vilket numret är.

```
scala> val f = Färg.Spader
scala> import Färg._
scala> Kort(Ruter, 7)
```

En annan fördelen med detta är att man lätt kan loopa från 1 till 4 för att gå igenom alla färger.

```
scala> val kortlek = for (f <- 1 to 4; v <- 1 to 13) yield Kort(f, v)
```

Nackdelen är att kompilatorn vid kompileringstid inte kollar om variablerna av misstag råkar ges något värde utanför det giltiga intervallet, t.ex. 42. Detta får vi själv hålla koll på vid körtid, till exempel med funktionen `require` eller **`if`**-satser, etc.

Istället kan man använda case-objekt enligt nedan deluppgifter och få hjälp av kompilatorn att hitta eventuella fel vid kompileringstid. Ett case-objekt är som ett vanligt singelton-objekt men det får automatiskt en toString samma som namnet och kan användas i matchningar etc. (mer om match i kaptitel **??**).

a) Deklarera följande uppräknade värden som singelton objekt med gemensam bastyp i en editor och klistra in i REPL med kommandot `:paste`. Med nyckelordet **`sealed`** så ”förseglas” klassen och inga andra direkta subtyper tillåts förutom de som finns i samma kod-fil eller block. I detta exempel med kortfärger vet vi ju att det inte finns fler än dessa fyra färger.

```
sealed trait Färg
case object Spader extends Färg
case object Hjärter extends Färg
case object Ruter extends Färg
case object Klöver extends Färg
```

Dessa kan sedan användas som parametrar till denna case-klass vid skapande av kortobjekt:

```
case class Kort(färg: Färg, valör: Int)
```

Skapa därefter några exempelinstanser av klassen `Kort`. Vad är fördelen med ovan angreppssätt jämfört med att använda heltalskonstanter?

b) Om man vill kunna iterera över alla värden är det bra om de finns i en samling med alla värden. Vi kan lägga en sådan i ett kompanjonsobjekt till bastypen. Uppdatera koden enligt nedan och klistra in på nytt i REPL med kommandot `:paste`. Skriv ut alla färgvärden med en **`for`**-sats.

```
sealed trait Färg
object Färg {
  val values = Vector(Spader, Hjärter, Ruter, Klöver)
}
```

```
case object Spader extends Färg
case object Hjärter extends Färg
case object Ruter extends Färg
case object Klöver extends Färg
```

Skapa en kortlek med 52 olika kort och blanda den sedan med `Random.shuffle` enligt nedan. Använd en dubbel **`for`**-sats och **`yield`**.

```
scala> val kortlek: Vector[Kort] = ???
scala> val blandad = scala.util.Random.shuffle(kortlek)
```

c) Skriv en funktion `def blandadKortlek: Vector[Kort] = ???` som ger en ny blandad kortlek varje gång den anropas med metoden i föregående uppgift.

d) Om man även vill ha en heltalsrepresentation med en medlem `toInt` för alla värden, kan man ge bastypen en parameter och i stället för en trait (som inte kan ha några parametrar) använda en abstrakt klass.

```
sealed abstract class Färg(final val toInt: Int)
object Färg {
  val values = Vector(Spader, Hjärter, Ruter, Klöver)
}
case object Spader  extends Färg(0)
case object Hjärter extends Färg(1)
case object Ruter   extends Färg(2)
case object Klöver  extends Färg(3)
```

Skapa en funktion `färgPoäng` som räknar ut summan av heltalsrepresentationen av alla färger hos en vektor med kort, och använd den sedan för att räkna ut `färgPoäng` för de första fem korten.

```
scala> def blandadKortlek: Vector[Kort] = ???
scala> def färgPoäng(xs: Vector[Kort]): Int = ???
scala> färgPoäng(blandadKortlek.take(5))
```

7.2 Extrauppgifter

Uppgift 12. Det visar sig att vår flitige ornitolog från uppgift 4 på sidan 83 sov på en av föreläsningarna i zoologi när hen var nolla på Natfak, och därför helt missat fylekategorin `Pjodd`. Hjälp vår stackars ornitolog så att fylehierarkin nu även omfattar Pjoddar. En Pjodd kan flyga som en Kråga men den `ÄrLiten` medan en Kråga `ÄrStor`. En Pjodd kvittrar dubbelt så många gånger som en Ånka kvackar. En Pjodd `KanKanskeSimma` där simkunnighetssannolikheten är 0.2. Låt ornitologen ånyo finna 42 slumpmässiga fåglar i skogen och filtrera fram lämpliga arter. Undersök sedan hur dessa väsnas, i likhet med deluppgift 4b.

7.3 Fördjupningsuppgifter

Uppgift 13. Hitta på en egen fördjupningsuppgift inspirerat av denna artikel på Stackoverflow: http://stackoverflow.com/questions/16173477/usages-of-null-nothing-unit-in-scala

Uppgift 14. Studera den djupa arvshierarkin i paketet `numbers` nedan som modellerar olika sorters tal i matematiken. Du kan även ladda ner koden här: github.com/lunduniversity/introprog/blob/master/compendium/examples/numbers.scala
Notera metoden `reduce` som reducerar ett tal till sin enklaste form och dess implementation överskuggas på lämpliga ställen med relevant reduktion.

a) Skriv kod som använder de olika konkreta klasserna i `package numbers`. Om du kompilerar koden i samma bibliotek som du kör igång REPL är det bara att använda paketet direkt:

```
$ scalac numbers.scala
$ scala
scala> numbers.  // Tryck Tab
AbstractComplex   AbstractNatural    AbstractReal   Frac    Nat      Polar
AbstractInteger   AbstractRational   Complex        Integ   Number   Real

scala> numbers.Integ(12)
res0: numbers.Integ = Integ(12)

scala> import numbers.Syntax._
import numbers.Syntax._

scala> 42.j
res1: numbers.Complex = Complex(Real(0),Real(42))

scala> 42.42.j
res2: numbers.Complex = Complex(Real(0),Real(42.42))
```

b) Andra på metoden + i `trait Number` så att den blir abstrakt och implementera den i alla konkreta klasser.

c) Implementera fler räknesätt och bygg vidare på koden så som du finner intressant.

d) Gör så att metoden `reduce` i klassen `AbstractRational` använder algoritmen Greatest Common Divisor (GCD)[13] så som beskrivs här:
www.artima.com/pins1ed/functional-objects.html#6.8
så att täljare och nämnare blir så små som möjligt.

```
package numbers

trait Number {
  def reduce: Number = this
  def isZero: Boolean
  def isOne: Boolean
  def +(that: Number): Number = ???
}
```

[13] https://sv.wikipedia.org/wiki/St%C3%B6rsta_gemensamma_delare

```
9
10 object Number {
11   def Zero = Nat(0)
12   def One  = Nat(1)
13   def Im(im: BigDecimal) = Complex(Real(0), Real(im))
14   def Im(im: Real)       = Complex(Real(0), im)
15   def j                  = Complex(Real(0), Real(1))
16   def Re(re: BigDecimal) = Complex(Real(re), Real(0))
17   def Re(re: Real)       = Complex(re, Real(0))
18 }
19
20 trait AbstractComplex extends Number {
21   def re: AbstractReal
22   def im: AbstractReal
23   def abs = Real(math.hypot(re.decimal.toDouble,im.decimal.toDouble))
24   def fi =  Real(math.atan2(im.decimal.toDouble, re.decimal.toDouble))
25   override def isZero: Boolean = re.decimal == 0 && im.decimal == 0
26   override def isOne: Boolean = abs.decimal == 1.0
27   override def reduce: AbstractComplex = if (im.decimal == 0) re.reduce else this
28 }
29
30 case class Complex(re: Real, im: Real) extends AbstractComplex
31
32 object Complex {
33   def apply(re: BigDecimal, im: BigDecimal) = new Complex(Real(re), Real(im))
34 }
35
36 case class Polar(
37     override val abs: Real,
38     override val fi: Real
39   ) extends AbstractComplex {
40   override def re = Real(abs.decimal.toDouble * math.cos(fi.decimal.toDouble))
41   override def im = Real(abs.decimal.toDouble * math.sin(fi.decimal.toDouble))
42 }
43
44 object Polar {
45   def apply(abs: BigDecimal, fi: BigDecimal) = new Polar(Real(abs), Real(fi))
46 }
47
48
49 trait AbstractReal extends AbstractComplex {
50   def decimal: BigDecimal
51   override def isZero = decimal == 0
52   override def isOne = decimal == 1
53   override def re = this
54   override def im = Number.Zero
55   override def reduce:  AbstractReal =
56     if (decimal == 0) Number.Zero else if (decimal == 1) Number.One else this
57 }
58
59 case class Real(decimal: BigDecimal) extends AbstractReal
60
61 trait AbstractRational extends AbstractReal {
62   def numerator: AbstractInteger
63   def denominator: AbstractInteger
64   override def decimal = BigDecimal(numerator.integ)
65   override def isOne = numerator.integ == denominator.integ
66   override def reduce: AbstractRational =
67     if (denominator.isOne) numerator.reduce else this // should use GCD
68 }
69
```

```
70  case class Frac(numerator: Integ, denominator: Integ) extends AbstractRational {
71    require(denominator.integ != 0, "denominator must be non-zero")
72  }
73
74  object Frac {
75    def apply(n: BigInt, d: BigInt) = new Frac(Integ(n), Integ(d))
76  }
77
78  trait AbstractInteger extends AbstractRational {
79    def integ: BigInt
80    override def numerator = this
81    override def denominator = Number.One
82    override def isZero = integ == 0
83    override def isOne = integ == 1
84    override def decimal: BigDecimal = BigDecimal(integ)
85    override def reduce: AbstractInteger =
86      if (isZero) Number.Zero else if (isOne) Number.One else this
87  }
88
89  case class Integ(integ: BigInt) extends AbstractInteger
90
91  trait AbstractNatural extends AbstractInteger
92
93  case class Nat(integ: BigInt) extends AbstractNatural{
94    require(integ >= 0, "natural numnbers must be non-negative")
95  }
96
97  object Syntax {
98    implicit class IntDecorator(i: Int){ def j = Number.Im(i) }
99    implicit class DoubleDecorator(d: Double){ def j = Number.Im(d) }
100 }
```

8. Övning: matching

Mål

- ☐ Kunna skapa och använda **match**-uttryck med konstanta värden, garder och mönstermatchning med case-klasser.
- ☐ Kunna skapa och använda case-objekt för matchningar på uppräknade värden.
- ☐ Känna till betydelsen av små och stora begynnelsebokstäver i case-grenar i en matchning, samt förstå hur namn binds till värden in en case-gren.
- ☐ Kunna hantera saknade värden med hjälp av typen `Option` och mänstermatchning på `Some` och `None`.
- ☐ Känna till hur metoden `unapply` används vid mönstermatchning.
- ☐ Känna till nyckelordet **sealed** och förstå nyttan med förseglade typer.
- ☐ Känna till **switch**-satser i Java.
- ☐ Känna till **null**.
- ☐ Kunna fånga undantag med **try-catch** och `scala.util.Try`.
- ☐ Känna till skillnaderna mellan **try-catch** i Scala och java.
- ☐ Kunna implementera `equals` med hjälp av en **match**-sats, som fungerar för finala klasser utan arv.
- ☐ Känna till relationen mellan `hashcode` och `equals`.
- ☐ Kunna använda `flatMap` tillsammans med `Option` och `Try`. ???
- ☐ Kunna skapa partiella funktioner med case-uttryck. ???

Förberedelser

- ☐ Studera begreppen i kapitel **??**.

8.1 Grunduppgifter

Uppgift 1. *Hur fungerar en **switch**-sats i Java (och flera andra språk)?* Det händer ofta att man vill testa om ett värde är ett av många olika alternativ. Då kan man använda en sekvens av många **if-else**, ett för varje alternativ. Men det finns ett annat sätt i Java och många andra språk: man kan använda **switch** som kollar flera alternativ i en och samma sats, se t.ex. en.wikipedia.org/wiki/Switch_statement.

a) Skriv in nedan kod i en kodeditor. Spara med namnet `Switch.java` och kompilera filen med kommandot `javac Switch.java`. Kör den med `java Switch` och ange din favoritgrönsak som argument till programmet. Vad händer? Förklara hur **switch**-satsen fungerar.

```
1 public class Switch {
2     public static void main(String[] args) {
3         String favorite = "selleri";
4         if (args.length > 0) {
5             favorite = args[0];
6         }
7         System.out.println("Din favoritgrönsak: " + favorite);
```

```
8         char firstChar = Character.toLowerCase(favorite.charAt(0));
9         System.out.print("Jag tycker att ");
10        switch (firstChar) {
11        case 'g':
12            System.out.println("gurka är gott!");
13            break;
14        case 't':
15            System.out.println("tomat är gott!");
16            break;
17        case 'b':
18            System.out.println("broccoli är gott!");
19            break;
20        default:
21            System.out.println(favorite + " är mindre gott...");
22            break;
23        }
24    }
25 }
```

b) Vad händer om du tar bort **break**-satsen på rad 16?

Uppgift 2. *Matcha på konstanta värden.* I Scala finns ingen **switch**-sats. I stället har Scala ett **match**-uttryck som är mer kraftfullt. Dock saknar Scala nyckelordet **break** och Scalas **match**-uttryck kan inte ”falla igenom” som skedde i uppgift 1b.

a) Skriv nedan program med en kodeditor och spara i filen `Match.scala`. Kompilera med `scalac Match.scala`. Kör med `scala Match` och ge som argument din favoritgrönsak. Vad händer? Förklara hur ett **match**-uttryck fungerar.

```
1  object Match {
2    def main(args: Array[String]): Unit = {
3      val favorite = if (args.length > 0) args(0) else "selleri"
4      println("Din favoritgrönsak: " + favorite)
5      val firstChar = favorite.toLowerCase.charAt(0)
6      val meThink = firstChar match {
7        case 'g' => "gurka är gott!"
8        case 't' => "tomat är gott!"
9        case 'b' => "broccoli är gott!"
10       case  _  =>  s"$favorite är mindre gott..."
11     }
12     println(s"Jag tycker att $meThink")
13   }
14 }
```

b) Vad blir det för felmeddelande om du tar bort case-grenen för defaultvärden och indata väljs så att inga case-grenar matchar? Är det ett exekveringsfel eller ett kompileringsfel?

c) Beskriv några skillnader i syntax och semantik mellan Javas flervalssats **switch** och Scalas flervalsuttryck **match**.

Uppgift 3. *Gard i case-grenar.* Med hjälp en gard (eng. *guard*) i en case-gren kan man begränsa med ett villkor om grenen ska väljas.

Utgå från koden i uppgift 2a och byt ut case-grenen för `'g'`-matchning till nedan variant med en gard med nyckelordet **`if`** (notera att det inte behövs parenteser runt villkoret):

```
    case 'g' if math.random > 0.5 => "gurka är gott ibland..."
```

Kompilera om och kör programmet upprepade gånger med olika indata tills alla grenar i **`match`**-uttrycket har exekverats. Förklara vad som händer.

Uppgift 4. *Mönstermatcha på attributen i case-klasser.* Scalas **`match`**-uttryck är extra kraftfulla om de används tillsammans med **`case`**-klasser: då kan attribut extraheras automatiskt och bindas till lokala variabler direkt i case-grenen som nedan exempel visar (notera att `v` och `rutten` inte behöver deklareras explicit). Detta kallas för **mönstermatchning**.

a) Vad skrivs ut nedan? Varför? Prova att byta namn på `v` och `rutten`.

```
scala> case class Gurka(vikt: Int, ärRutten: Boolean)
scala> val g = Gurka(100, true)
scala> g match { case Gurka(v,rutten) => println("G" + v + rutten) }
```

b) Skriv sedan nedan i REPL och tryck TAB två gånger efter punkten. Vad har `unapply`-metoden för resultattyp?

```
scala> Gurka.unapply   // Tryck TAB två gånger
```

> *Bakgrund för kännedom:* Case-klasser får av kompilatorn automatiskt ett kompanjonsobjekt (eng. *companion object*), i detta fallet **`object`** `Gurka`. Det objektet får av kompilatorn automatiskt en `unapply`-metod. Det är `unapply` som anropas ”under huven” när case-klassernas attribut extraheras vid mönstermatchning, men detta sker alltså automatiskt och man behöver inte explicit nyttja `unapply` om man inte själv vill implementera s.k. extraherare (eng. *extractors*); om du är nyfiken på detta, se fördjupningsuppgift 21.

c) Anropa `unapply`-metoden enligt nedan. Vad blir resultatet?

```
scala> Gurka.unapply(g)
```

Vi ska i senare uppgifter undersöka hur typerna `Option` och `Some` fungerar och hur man kan ha nytta av dessa i andra sammanhang.

d) Spara programmet nedan i filen `vegomatch.scala` och kompilera med `scalac vegomatch.scala` och kör med `scala vegomatch.Main 1000` i terminalen. Förklara hur predikatet `ärÄtvärd` fungerar.

```
package vegomatch

trait Grönsak {
  def vikt: Int
  def ärRutten: Boolean
}

case class Gurka(vikt: Int, ärRutten: Boolean) extends Grönsak
case class Tomat(vikt: Int, ärRutten: Boolean) extends Grönsak
```

```
10
11 object Main {
12   def slumpvikt:    Int     = (math.random * 420 + 42).toInt
13   def slumprutten:  Boolean = math.random > 0.8
14   def slumpgurka:   Gurka   = Gurka(slumpvikt, slumprutten)
15   def slumptomat:   Tomat   = Tomat(slumpvikt, slumprutten)
16   def slumpgrönsak: Grönsak =
17     if (math.random > 0.2) slumpgurka else slumptomat
18
19   def ärÄtvärd(g: Grönsak): Boolean = g match {
20     case Gurka(v, rutten) if v > 100 && !rutten => true
21     case Tomat(v, rutten) if v >  50 && !rutten => true
22     case _ => false
23   }
24
25   def main(args: Array[String]): Unit = {
26     val skörd = Vector.fill(args(0).toInt)(slumpgrönsak)
27     val ätvärda = skörd.filter(ärÄtvärd)
28     println("Antal skördade grönsaker: " + skörd.size)
29     println("Antal ätvärda grönsaker:  " + ätvärda.size)
30   }
31 }
```

Uppgift 5. Man kan åstadkomma urskiljningen av de ätbara grönsakerna i uppgift 4 med polymorfism i stället för `match`.

a) Gör en ny variant av ditt program enligt nedan riktlinjer och spara den modifierade koden i filen `vegopoly.scala` och kompilera och kör.

- Ta bort predikatet `ärÄtvärd` i objektet `Main` och inför i stället en abstrakt metod `def ärÄtbar: Boolean` i traiten `Grönsak`.
- Inför konkreta `val`-medlemmar i respektive grönsak som definierar ätbarheten.
- Ändra i huvudprogrammet i enlighet med ovan ändringar så att `ärÄtvärd` anropas som en metod på de skördade grönsaksobjekten när de ätvärda ska filtreras ut.

b) Lägg till en ny grönsak `case class Broccoli` och definiera dess ätbarhet. Ändra i slump-funktionerna så att broccoli blir ovanligare än gurka.

c) Jämför lösningen med `match` i uppgift 4 och lösningen ovan med polymorfism. Vilka är för- och nackdelarna med respektive lösning. Diskutera två olika situationer på ett hypotetiskt företag som utvecklar mjukvara för jordbrukssektorn: 1) att uppsättningen grönsaker inte ändras särskilt ofta medan definitionerna av ätbarhet ändras väldigt ofta och 2) att uppsättningen grönsaker ändras väldigt ofta men att ätbarhetsdefinitionerna inte ändras särskilt ofta.

Uppgift 6. *Matcha på case-objekt och nyttan med `sealed`.* Skapa nedan kod i en editor, och klistra in i REPL med kommandot `:pa`. Notera nyckelordet `sealed` som används för att försegla en typ. En **förseglad typ** måste ha alla sina subtyper i en och samma kodfil.

```
sealed trait Färg
object Färg {
  val values = Vector(Spader, Hjärter, Ruter, Klöver)
}
case object Spader  extends Färg
case object Hjärter extends Färg
case object Ruter   extends Färg
case object Klöver  extends Färg
```

a) Skapa en funktion `def parafärg(f: Färg): Färg` i en editor, som med hjälp av ett match-uttryck returnerar parallellfärgen till en färg. Parallellfärgen till `Hjärter` är `Ruter` och vice versa, medan parallellfärgen till `Klöver` är `Spader` och vice versa. Klistra in funktionen i REPL.

```
scala> parafärg(Spader)
scala> val xs = Vector.fill(5)(Färg.values((math.random * 4).toInt))
scala> xs.map(parafärg)
```

b) Vi ska nu undersöka vad som händer om man glömmer en av casegrenarna i matchningen i `parafärg`? ”Glöm” alltså avsiktligt en av casegrenarna och klistra in den nya `parafärg` med den ofullständiga matchningen. Hur lyder varningen? Kommer varningen vid körtid eller vid kompilering?

c) Anropa `parafärg` med den ”glömda” färgen. Hur lyder felmeddelandet? Är det ett kompileringsfel eller ett körtidsfel?

d) Förklara vad nyckelordet **`sealed`** innebär och vilken nytta man kan ha av att **försegla** en supertyp.

Uppgift 7. *Betydelsen av små och stora begynnelsebokstäver vid matchning.* För att åstadkomma att namn kan bindas till variabler vid matchning utan att de behöver deklareras i förväg (som vi såg i uppgift 4a) så har identifierare med liten begynnelsebokstav fått speciell betydelse: den tolkas av kompilatorn som att du vill att en variabel binds till ett värde vid matchningen. En identifierare med stor begynnelsebokstav tolkas däremot som ett konstant värde (t.ex. ett case-objekt eller ett case-klass-mönster).

a) *En case-gren som fångar allt*. En case-gren med en identifierare med liten begynnelsebokstav som saknar gard kommer att matcha allt. Prova nedan i REPL, men försök lista ut i förväg vad som kommer att hända. Vad händer?

```
scala> val x = "urka"
scala> x match {
         case str if str.startsWith("g") => println("kanske gurka")
         case vadsomhelst => println("ej gurka: " + vadsomhelst)
       }
scala> val g = "gurka"
scala> g match {
         case str if str.startsWith("g") => println("kanske gurka")
         case vadsomhelst => println("ej gurka: " + vadsomhelst)
       }
```

b) *Fallgrop med små begynnelsbokstäver.* Innan du provar nedan i REPL, försök gissa vad som kommer att hända. Vad händer? Hur lyder varningarna och vad innebär de?

```
scala> val any: Any = "gurka"
scala> case object Gurka
scala> case object tomat
scala> any match {
         case Gurka => println("gurka")
         case tomat => println("tomat")
         case _ => println("allt annat")
       }
```

c) *Använd backticks för att tvinga fram match på konstant värde.* Det finns en utväg om man inte vill att kompilatorn ska skapa en ny lokal variabel: använd specialtecknet *backtick*, som skrivs ` och kräver speciella tangentbordstryck.[14] Gör om föregående uppgift men omgärda nu identifieraren `tomat` i tomat-case-grenen med backticks, så här: `case` `` `tomat` `` `=>` `...`

Uppgift 8. *Använda `Option` och matcha på värden som kanske saknas.* Man behöver ofta skriva kod för att hantera värden som eventuellt saknas, t.ex. saknade telefonnummer i en persondatabas. Denna situation är så pass vanlig att många språk har speciellt stöd för saknande värden.

I Java[15] används värdet `null` för att indikera att en referens saknar värde. Man får då komma ihåg att testa om värdet saknas varje gång sådana värden ska behandlas, , t.ex. med `if (ref != null) { ...} else { ... }`. Ett annat vanligt trick är att låta `-1` indikera saknade positiva heltal, till exempel saknade index, som får behandlas med `if (i != -1) { ...} else { ... }`.

I Scala finns en speciell typ `Option` som möjliggör smidig och typsäker hantering av saknade värden. Om ett kanske saknat värde packas in i en `Option` (eng. *wrapped in an Option*), finns det i en speciell slags samling som bara kan innehålla *inget* eller *något* värde, och alltså har antingen storleken `0` eller `1`.

a) Förklara vad som händer nedan.

```
scala> var kanske: Option[Int] = None
scala> kanske.size
scala> kanske = Some(42)
scala> kanske.size
scala> kanske.isEmpty
scala> kanske.isDefined
scala> def ökaOmFinns(opt: Option[Int]): Option[Int] = opt match {
         case Some(i) => Some(i + 1)
         case None    => None
       }
scala> val annanKanske = ökaOmFinns(kanske)
scala> def öka(i: Int) = i + 1
scala> val merKanske = kanske.map(öka)
```

[14] Fråga någon om du inte hittar hur man gör backtick ` på ditt tangentbord.

[15] Scala har också `null` men det behövs bara vid samverkan med Java-kod.

b) Mönstermatchingen ovan är minst lika knölig som en `if`-sats, men tack vare att en `Option` är en slags (liten) samling finns det smidigare sätt. Förklara vad som händer nedan.

```
val meningen = Some(42)
val ejMeningen = Option.empty[Int]
meningen.map(_ + 1)
ejMeningen.map(_ + 1)
ejMeningen.map(_ + 1).orElse(Some("saknas")).foreach(println)
meningen.map(_ + 1).orElse(Some("saknas")).foreach(println)
```

c) *Samlingsmetoder som ger en `Option`.* Förklara för varje rad nedan vad som händer. En av raderna ger ett felmeddelande; vilken rad och vilket felmeddelande?

```
val xs = (42 to 84 by 5).toVector
val e = Vector.empty[Int]
xs.headOption
xs.headOption.get
xs.headOption.getOrElse(0)
xs.headOption.orElse(Some(0))
e.headOption
e.headOption.get
e.headOption.getOrElse(0)
e.headOption.orElse(Some(0))
Vector(xs, e, e, e)
Vector(xs, e, e, e).map(_.lastOption)
Vector(xs, e, e, e).map(_.lastOption).flatten
xs.lift(0)
xs.lift(1000)
e.lift(1000).getOrElse(0)
xs.find(_ > 50)
xs.find(_ < 42)
e.find(_ > 42).foreach(_ => println("HITTAT!"))
```

d) Vilka är fördelerna med `Option` jämfört med `null` eller `-1` om man i sin kod glömmer hantera saknade värden?

Uppgift 9. *Kasta undantag.* Om man vill signalera att ett fel eller en onormal situtation uppstått så kan man **kasta** (eng. *throw*) ett **undantag** (eng. *exception*). Då avbryts programmet direkt med ett felmeddelande, om man inte väljer att **fånga** (eng. *catch*) undantaget.

a) Vad händer nedan?

```
scala> throw new Exception("PANG!")
scala> java.lang.   // Tryck TAB efter punkten
scala> throw new IllegalArgumentException("fel fel fel")
scala> val carola = try {
         throw new Exception("stormvind!")
         42
       } catch { case e: Throwable => println("Fångad av en " + e); -1 }
```

b) Nämn ett par undantag som finns i paketet `java.lang` som du kan gissa vad de innebär och i vilka situationer de kastas.

c) Vilken typ har variabeln `carola` ovan? Vad hade typen blivit om catch-grenen hade returnerat en sträng i stället?

Uppgift 10. *Fånga undantantag i Java med en* ***`try-catch`****-sats.* Det finns som vi såg i förra uppgiften inbyggt stöd i JVM för att hantera när program avbryts på oväntade sätt, t.ex. på grund av division med noll eller ej förväntade indata från användaren. Skriv in nedan Java-program i en editor och spara i en fil med namnet `TryCatch.java` och kompilera med `javac TryCatch.java` i terminalen.

```
1  // TryCatch.java
2
3  public class TryCatch {
4      public static void main(String[] args) {
5          int input;
6          int output;
7          if (args[0].equals("safe")) {
8              try {
9                  input = Integer.parseInt(args[1]);
10                 System.out.println("Skyddad division!");
11                 output = 42 / input;
12             } catch (Exception e) {
13                 System.out.println("Undantag fångat: " + e);
14                 System.out.println("Dividerar ändå med säker default!");
15                 input = 1;
16                 output = 42 / input;
17             }
18         } else {
19             input = Integer.parseInt(args[0]);
20             System.out.println("Oskyddad division!");
21             output = 42 / input;
22         }
23         System.out.println("42 / " + input + " == " + output);
24     }
25 }
```

a) Förklara vad som händer när du kör programmet med olika indata:

```
1 $ java TryCatch 42
2 $ java TryCatch 0
3 $ java TryCatch safe 42
4 $ java TryCatch safe 0
5 $ java TryCatch
```

b) Vad händer om du ”glömmer bort” raden 15 och därmed missar att initialisera input? Hur lyder felmeddelandet? Är det ett körtidsfel eller kompileringsfel?

c) Beskriv några skillnader och likheter i syntax och semantik mellan **`try-catch`** i Java respektive Scala.

Uppgift 11. *Fånga undantantag i Scala med* `scala.util.Try`. I paketet `scala.util` finns typen `Try` med stort T som är som en slags samling som kan innehålla antingen ett ”lyckat” eller ”misslyckat” värde. Om beräkningen av värdet lyckades och inga undantag kastas blir värdet inkapslat i en `Success`, annars blir undantaget inkapslat i en `Failure`. Man kan extrahera värdet, respektive undantaget, med mönstermatchning, men det är oftast smidigare att använda samlingsmetoderna `map` och `foreach`, i likhet med hur `Option` används. Det finns även en smidig metod `recover` på objekt av typen `Try` där man kan skicka med kod som körs om det uppstår en undantagssituation.

a) Förklara vad som händer nedan.

```
scala> def pang = throw new Exception("PANG!")
scala> import scala.util.{Try, Success, Failure}
scala> Try{pang}
scala> Try{pang}.recover{case e: Throwable => s"desarmerad bomb: $e"}
scala> Try{"tyst"}.recover{case e: Throwable => s"desarmerad bomb: $e"}
scala> def kanskePang = if (math.random > 0.5) "tyst" else pang
scala> def kanskeOk = Try{ kanskePang}
scala> val xs = Vector.fill(100)(kanskeOk)
scala> xs(13) match {
         case Success(x) => ":)"
         case Failure(e) => ":( " + e
       }
scala> x(13).isSuccess
scala> x(13).isFailure
scala> xs.count(_.isFailure)
scala> xs.find(_.isFailure)
scala> val badOpt = xs.find(_.isFailure)
scala> val goodOpt = xs.find(_.isSuccess)
scala> badOpt
scala> badOpt.get
scala> badOpt.get.get
scala> badOpt.map(_.getOrElse("bomben desarmerad!")).get
scala> goodOpt.map(_.getOrElse("bomben desarmerad!")).get
scala> xs.map(_.getOrElse("bomben desarmerad!")).foreach(println)
scala> xs.map(_.toOption)
scala> xs.map(_.toOption).flatten
scala> xs.map(_.toOption).flatten.size
```

b) Vad har funktionen `pang` för returtyp?

c) Varför får funktionen `kanskePang` den härledda returtypen `String`?

Uppgift 12. *Metoden* `equals`. Om man överskuggar den befintliga metoden `equals` så kommer metoden == att fungera annorlunda. Man kan då själv åstadkomma innehållslikhet i stället för referenslikhet. Vi börjar att studera den befintliga equals med referenslikhet.

a) Vad händer nedan? Om du trycker TAB *två* gånger efter ett metodnamn får du se metodens signatur. Vilken signatur har metoden `equals`?

```
scala> class Gurka(val vikt: Int, ärÄtbar: Boolean)
scala> val g1 = new Gurka(42, true)
scala> val g2 = g1
```

```
4 scala> val g3 = new Gurka(42, true)
5 scala> g1 == g2
6 scala> g1 == g3
7 scala> g1.equals  // tryck TAB två gånger
```

b) Rita minnessituationen efter rad 4. 

c) *Överskugga metoderna `equals` och `hashCode`.*

Bakgrund för kännedom: Det visar sig förvånande komplicerat att implementera innehållslikhet med metoden `equals` så att den ger bra resultat under alla speciella omständigheter. Till exempel måste man även överskugga en metod vid namn `haschCode` om man överskuggar `equals`, eftersom dessa båda används gemensamt av effektivitetsskäl för att skapa den interna lagringen av objekten i vissa samlingar. Om man missar det kan objekt bli ”osynliga” i `hashCode`-baserade samlingar – men mer om detta i senare kurser. Om objekten ingår i en öppen arvshierarki blir det också mer komplicerat; det är enklare om man har att göra med finala klasser. Dessutom krävs speciella hänsyn om klassen har en typparameter.

Definera klassen nedan i REPL med överskuggade `equals` och `hashCode`; den ärver inte något och är final.

```
// fungerar fint om klassen är final och inte ärver något
final class Gurka(val vikt: Int, ärÄtbar: Boolean) {
  override def equals(other: Any): Boolean = other match {
    case that: Gurka => this.vikt == that.vikt
    case _ => false
  }
  override def hashCode: Int = (vikt, ärÄtbar).## //förklaras sen
}
```

d) Vad händer nu nedan, där `Gurka` nu har en överskuggad `equals` med innehållslikhet?

```
1 scala> val g1 = new Gurka(42, true)
2 scala> val g2 = g1
3 scala> val g3 = new Gurka(42, true)
4 scala> g1 == g2
5 scala> g1 == g3
```

e) Hur märker man ovan att den överskuggade `equals` medför att == nu ger innehållslikhet? Jämför med deluppgift a. 

I uppgift 19 får du prova på att följa det fullständiga receptet i 8 steg för att överskugga en `equals` enligt konstens alla regler. I efterföljande kurs kommer mer träning i att hantera innehållslikhet och hash-koder. I Scala får man ett objekts hash-kod med metoden ##.[16]

[16] Om du är nyfiken på hash-koder, läs mer här: en.wikipedia.org/wiki/Java_hashCode().

8.2 Extrauppgifter

Uppgift 13. *Polynom*. Med hjälp av koden nedan, kan man göra följande:

```
scala> :pa polynomial.scala

scala> import polynomial._

scala> Const(1) * x
res0: polynomial.Term = x

scala> (x*5)^2
res1: polynomial.Prod = 25x^2

scala> Poly(x*(-5), y^4, (z^2)*3)
res2: polynomial.Poly = -5x + y^4 + 3z^2
```

✎ a) Förklara vad som händer ovan genom att studera koden för `object polynomial` nedan i filen `polynomial.scala`.[17]

```
object polynomial {

  sealed trait Term {
    def *(that: Term): Term
  }

  case class Const(value: BigDecimal) extends Term {

    def toSilentString: String = this match {
      case Const.One      => ""
      case Const.MinusOne => "-"
      case _              => value.toString
    }

    override def toString = value.toString

    override def *(that: Term): Term = that match {
      case Const(d)    => Const(d * value)
      case v: Var => Prod(this, Set(v))
      case Prod(c, vs) => Prod(Const(c.value * value), vs)
    }

    def *(d: BigDecimal): Const = Const(d * value)

    def ^(e: Int): Const = Const(value.pow(e))

  }

  object Const {
    final val Zero     = Const(BigDecimal(0))
    final val One      = Const(BigDecimal(1))
    final val MinusOne = Const(BigDecimal(-1))
```

[17] Koden finns även här:
github.com/lunduniversity/introprog/tree/master/compendium/examples/polynomial

```
33  }
34
35  case class Var(name: Char, exp: Int = 1) extends Term {
36
37    private def silentExpString: String =
38      if (exp == 1) "" else "^"+exp.toString
39
40    override def toString = s"$name$silentExpString"
41
42    def ^(e: Int): Var = Var(name, e * exp)
43
44    def *(c: BigDecimal) = Prod(Const(c), Set(this))
45
46    override def *(that: Term): Term = that match {
47      case c: Const => Prod(c, Set(this))
48
49      case v: Var =>
50        if (v.name == name) Var(name, v.exp + exp)
51        else Prod(Const.One, Set(this, v))
52
53      case p: Prod => p * this
54    }
55
56  }
57
58  object Var{
59
60    def apply(d: BigDecimal, name: Char): Prod =
61      Prod(Const(d), Set(Var(name)))
62
63    def apply(d: BigDecimal, name: Char, exp: Int): Prod =
64      Prod(Const(d), Set(Var(name, exp)))
65
66    def addExp(v1: Var, v2: Var): Var = Var(v1.name, v1.exp + v2.exp)
67
68    def multiply(v1: Var, vs: Set[Var]): Set[Var] = {
69      if (!vs.contains(v1)) vs + v1
70      else vs.map(v2 => if (v1.name == v2.name) addExp(v1, v2) else v2)
71    }
72
73    def multiply(vs1: Set[Var], vs2: Set[Var]): Set[Var] = {
74      var result = vs2
75      vs1.foreach{ v1 => result = multiply(v1, result) }
76      result
77    }
78
79  }
80
81  case class Prod(const: Const, vars: Set[Var]) extends Term  {
82
83    override def toString = s"${const.toSilentString}${vars.mkString}"
84
85    override def *(that: Term): Term = that match {
86      case Const(d) => Prod(Const(d * const.value), vars)
87
88      case v: Var => Prod(const, Var.multiply(v, vars))
```

```
89
90        case Prod(Const(d), vs)  =>
91          Prod(Const(const.value * d), Var.multiply(vs, vars))
92      }
93
94      def ^(e: Int) = Prod(const ^ e, vars.map(_ ^ e))
95    }
96
97    case class Poly(xs: Set[Term]) {
98      override def toString = xs.mkString(" + ")
99    }
100
101   object Poly {
102     def apply(ts: Term*) : Poly = Poly(ts.toSet)
103   }
104
105   val (x, y, z, s, t) = (Var('x'), Var('y'), Var('z'), Var('s'), Var('t'))
106
107 }
```

b) Bygg vidare på `object polynomial` och implementera addition mellan olika termer.

✎ **Uppgift 14.** Studera dokumentationen för `Option` här och se om du känner igen några av metoderna som också finns på samlingen `Vector`:
www.scala-lang.org/api/current/index.html#scala.Option
Förklara hur metoden `contains` på en `Option` fungerar med hjälp av dokumentationens exempel.

Uppgift 15. Gör motsvarande program i Scala som visas i uppgift 10, men utnyttja att Scalas `try-catch` är ett uttryck. Kompilera och kör och testa så att de ur användarens synvinkel fungerar precis på samma sätt. Notera de viktigaste skillnaderna mellan de båda programmen.

8.3 Fördjupningsuppgifter

a) Bygg vidare på `object polynomial` och implementera metoden `def reduce: Poly` i case-klassen `Poly` som förenklar polynom om flera Prod-termer kan adderas.

Uppgift 16. TODO!!! Speciella matchningar. @ och _*

Uppgift 17. TODO!!! *Implementera en egen, typsäker innehållstest med metoden* ===. Ska detta vara med ???

Uppgift 18. *Vad är hashcode?* TODO!!! Undersöka ## i REPL och visa hur tokigt det kan bli om man inte överskuggar hash-kod.

Uppgift 19. *Överskugga `equals` med innehållslikhet även för icke-finala klasser.* Nedan visas delar av klassen `Complex` som representerar ett komplext tal med realdel och imaginärdel. I stället för att, som man ofta gör i Scala, använda

en case-klass och en `equals`-metod som automatiskt ger innehållslikhet, ska du träna på att implementera en egen `equals`.

```
class Complex(re: Double, im: Double) {
  def abs: Double = math.hypot(re, im)
  override def toString = s"Complex($re, $im)"
  def canEqual(other: Any): Boolean = ???
  override def hashCode: Int  = ???
  override def equals(other: Any): Boolean = ???
}
case object Complex {
  def apply(re: Double, im: Double): Complex = new Complex(re, im)
}
```

Följ detta **recept**[18] i 8 steg för att överskugga `equals` med innehållslikhet som fungerar även för klasser som inte är `final`:

1. Inför denna metod: `def canEqual(other: Any): Boolean`
   Observera att typen på parametern ska vara `Any`. Om detta görs i en subklass till en klass som redan implementerat `canEqual`, behövs även `override`.

2. Metoden `canEqual` ska ge `true` om `other` är av samma typ som `this`, alltså till exempel:
   `def canEqual(other: Any): Boolean = other.isInstanceOf[Complex]`

3. Inför metoden `equals` och var noga med att parametern har typen `Any`:
   `override def equals(other: Any): Boolean`

4. Implementera metoden `equals` med ett match-uttryck som börjar så här:
   `other match { ... }`

5. Match-uttrycket ska ha två grenar. Den första grenen ska ha ett typat mönster för den klass som ska jämföras:
   `case that: Complex =>`

6. Om du implementerar `equals` i den klass som inför `canEqual`, börja uttrycket med:
   `(that canEqual this) &&`
   och skapa därefter en fortsättning som baseras på innehållet i klassen, till exempel: `this.re == that.re && this.im == that.im`
   Om du överskuggar en *annan* equals än den standard-equals som finns i `AnyRef`, vill du förmodligen börja det logiska uttrycket med att anropa superklassens equals-metod: `super.equals(that) &&` men du får fundera noga på vad likhet av underklasser egentligen ska innebära i ditt speciella fall.

7. Den andra grenen i matchningen ska vara: `case _ => false`

[18] Detta recept bygger på http://www.artima.com/pins1ed/object-equality.html

8. Överskugga `hashCode`, till exempel genom att göra en tupel av innehållet i klassen och anropa metoden ## på tupeln så får du i en bra hashcode:
   `override def hashCode: Int  = (re, im).##`

Uppgift 20. TODO!!! Överskugga equals vid arv för Comlpex och Rational nedan

```
trait Number {
  override def equals(other: Any): Boolean = ???
}
class Complex(re: Double, im: Double) extends Number {
  override def equals(other: Any): Boolean = ???
}
class Rational(numerator: Int, denominator: Int) extends Number {
  override def equals(other: Any): Boolean = ???
}
```

Uppgift 21. TODO!!! Skapa din egen extraktor med metoden `unapply`.

Uppgift 22. TODO!!! *flatten och flatMap med Option och Try* Ska detta vara ordinarie uppgift eller fördjupning???

Uppgift 23. TODO!!! *partiella funktioner och metoderna collect och collectFirst på samlingar* Ska detta vara ordinarie uppgift eller fördjupning???

Uppgift 24. TODO!!! Plynomdivision ???

9. Övning: `matrices`

Mål

- ☐ Kunna skapa och använda matriser med nästlade strukturer av `Vector`.
- ☐ Kunna iterera över elementen i en matris med nästlade **`for`**-satser och **`for`**-**`yield`**-uttryck, samt nästlad applicering av `map` respektive `foreach`.
- ☐ Kunna skapa och använda funktioner som tar matriser som parametrar.
- ☐ Känna till generiska funktioner.
- ☐ Känna till generiska klasser.
- ☐ Kunna skapa och använda matriser med hjälp inbyggda arrayer i Java.
- ☐ Kunna anända nästlade **`for`**-satser i Java för att iterera över elementen i en matris.

Förberedelser

- ☐ Studera begreppen i kapitel **??**.

9.1 Grunduppgifter

Uppgift 1. *Skapa matriser med hjälp av nästlade samlingar.* Man kan i ett datorprogram, med hjälp av samlingar som innehåller samlingar, skapa nästlade strukturer som kan indexeras i två dimensioner och på så sätt representera en matematisk **matris**.[19]

Bakgrund för kännedom: En **matris** inom matematiken innehåller ett antal rader och kolumner (även kallade kolonner). I en matematisk matris har alla rader lika många element och även alla kolumner har lika många element. En matris av dimension $m \times n$ har $m \cdot n$ stycken element, där m är antalet rader och n är antalet kolumner. En matris $A_{m,n}$ av dimension $m \times n$ ritas ofta så här:

$$A_{m,n} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}$$

Exempel: En heltalsmatris $M_{2,5}$ av dimension 2×5 där element $m_{2,5} = 7$:

$$M = \begin{pmatrix} 5 & 2 & 42 & 4 & 5 \\ 3 & 4 & 18 & 6 & 7 \end{pmatrix}$$

a) Rita minnessituationen efter tilldelningen på rad 1 nedan. Vad har `m` för typ och värde? Vad har `m` för dimensioner? Hur sker indexeringen i ett datorprogram jämfört med i matematiken?

```
1 scala> val m = Vector((1 to 5).toVector, (3 to 7).toVector)
2 scala> m.apply(0).apply(1)
3 scala> m(1)
4 scala> m(1)(4)
```

[19] sv.wikipedia.org/wiki/Matris

b) Vad ger uttrycken på raderna 2, 3 och 4 ovan för värden och typ?

c) Man kan i ett datorprogram mycket väl skapa tvådimensionella, nästlade strukturer där raderna *inte* innehåller samma antal element. Det blir då ingen äkta matris i strikt matematisk mening, men man kallar ofta ändå en sådan struktur för en ”matris”. Vilken typ har variablerna `m2`, `m3`, `m4` och `m5` nedan?

```
1 scala> val m2 = Vector(Vector(1,2,3),Vector(4,5),Vector(42))
2 scala> val m3 = Vector(Vector(1,2), Vector(1.0, 2.0, 3.0))
3 scala> m3(1)
4 scala> val m4 = m3(1) +: Vector("a") +: m3
5 scala> val m5 = Vector.fill(42){ m2(1).map(e => (e * math.random).toInt) }
```

d) Rita minnessituationen efter tilldelingen av `m2` på rad 1 ovan.

e) Vilken av variablerna `m2`, `m3`, `m4` och `m5` ovan representerar en äkta matris i matematisk mening? Vilken är dess dimensioner?

Uppgift 2. *Skapa och itererar över matriser.* Vi ska skapa matriser där varje rad representerar 5 kast med en tärning spelet Yatzy.[20]

a) Definiera i REPL en funktion `def throwDie: Int = ???` som returnerar ett slumptal mellan 1 och 6.

b) Skapa nedan heltalsmatris i REPL. Vilken dimension får matrisen?

```
1 val ds1 = for (i <- 1 to 1000) yield {
2             for (j <- 1 to 5) yield throwDie
3           }
```

c) Man kan också använda nedan varianter för att skapa en heltalsmatris. Vilken av varianterna `ds1` ... `ds5` tycker du är lättast att läsa och förstå? Prova respektive variant i REPL och ange vilken typ på `ds1` ... `ds5` som härleds av kompilatorn.

```
1 val ds2 = (1 to 1000).map(i => (1 to 5).map(j => throwDie))
2 val ds3 = (1 to 1000).map(i => Vector.fill(5)(throwDie))
3 val ds4 = for (i <- 1 to 1000) yield Vector.fill(5)(throwDie)
4 val ds5 = Vector.fill(1000)(Vector.fill(5)(throwDie))
```

d) Definiera en funktion
`def roll(n: Int): Vector[Int] = ???`
som ger en heltalsvektor med n stycken slumpvisa tärningskast. Kasten ska vara sorterade i växande ordning; använd för detta ändamål samlingsmetoden `sorted`.

e) Definera i REPL en funktion `isYatzy(xs: Vector[Int]): Boolean = ???` som testar om alla elementen i en heltalsvektor är samma. Använd samlingsmetoden `forall`.

f) Implementera `isYatzy` igen med ett imperativt angreppssätt som använder en `while`-sats (alltså utan att använda funktionella `forall`). Ta hjälp av en variabel `i` som håller reda på index och en variabel `foundDiff` som håller reda

[20] sv.wikipedia.org/wiki/Yatzy

på om ett avvikande värde upptäcks. Funktionen blir ca 10 rader, så det kan vara lämpligt att öppna en editor att skriva i medan du klurar ut lösningen. Börja med att skriva pseudokod, gärna med penna på papper. Prova genom att klistra in i REPL.

g) Skapa en funktion
`def diceMatrix(m: Int, n: Int): Vector[Vector[Int]] = ???`
som med hjälp av funktionen `roll` skapar en matris med `m` st vektorer med vardera `n` slumpvisa tärningskast.

h) Skapa en funktion som returnerar en utskriftsvänlig sträng
`def diceMatrixToString(xss: Vector[Vector[Int]]): String = ???`
med hjälp av `map` och `mkString`, som fungerar enligt nedan.

```
scala> println(diceMatrixToString(diceMatrix(10, 5)))
4 5 5 3 3
1 4 1 3 1
1 3 1 5 5
6 4 4 5 5
2 1 5 6 5
1 2 2 3 6
1 3 2 4 5
2 2 3 2 2
2 6 3 4 6
4 5 5 2 3
```

i) Ett imperativt sätt[21] att göra detta på visas nedan. Förklara hur nedan kod fungerar. Vad händer om `xss` är tom? Vad händer om `xss` bara innehåller tomma vektorer? Nämn en fördel och en nackdel med att använda `val sb: StringBuilder` och `append`, jämfört med en vanlig `var s: String` och + för tillägg i slutet.

```
def diceMatrixToString(xss: Vector[Vector[Int]]): String = {
  val sb = new StringBuilder()
  for(m <- 0 until xss.size) {
    for(n <- 0 until xss(m).size) {
      sb.append(xss(m)(n))
      if (n < xss(m).size - 1) sb.append(" ")
      else if (m < xss.size - 1) sb.append("\n")
    }
  }
  sb.toString
}
```

j) Implementera funktionen
`def filterYatzy(xss: Vector[Vector[Int]]): Vector[Vector[Int]]`
som filtrerar fram alla yatzy-rader i matrisen `xss` enligt nedan. Använd din funktion `isYatzy` och samlingsmetoden `filter`.

[21] Imperativa anreppssätt är nödvändiga att kunna när du stöter på samlingar och/eller språk som saknar funktionsprogrammeringsmöjligheter med `map`, `mkString` etc.

```
scala> println(diceMatrixToString(filterYatzy(diceMatrix(10000, 5))))
2 2 2 2 2
3 3 3 3 3
1 1 1 1 1
3 3 3 3 3
4 4 4 4 4
6 6 6 6 6
2 2 2 2 2
3 3 3 3 3
2 2 2 2 2
6 6 6 6 6
4 4 4 4 4
2 2 2 2 2
4 4 4 4 4
```

k) Gör som träning en imperativ implementation av `filterYatzy` med en **`for`**-sats (alltså utan att använda `filter`, och utan att använda **`yield`**).

l) Tycker du din imperativa lösning är lättare eller svårare att läsa och förstå jämfört nedan funktionella lösning med ett **`for`**-uttryck och **`yield`**?

```
def filterYatzy(xss: Vector[Vector[Int]]): Vector[Vector[Int]] = {
  for (i <- 0 until xss.size if isYatzy(xss(i))) yield xss(i)
}.toVector
```

m) Implementera funktionen
`def yatzyPips(xss: Vector[Vector[Int]]): Vector[Int]`
som ger en vektor med tärningsvärdena för de kast i matrisen `xss` som gav yatzy enligt nedan. Använd din funktion `isYatzy` och samlingsmetoden `filter`.

```
scala> yatzyPips(diceMatrix(10000, 5))
res42: Vector[Int] = Vector(3, 5, 6, 6, 3, 3, 2, 6, 1, 3)
```

Uppgift 3. *Strängtabell med rubrikrad.* Denna övning utgör en början på det du ska göra under veckans laboration.

a) Implementera case-klassen `Table` enligt nedan specifikation. Du kan förutsätta att alla rader har lika många kolumner som antalet element i `headings`, samt att alla rubrikerna i `headings` är unika. Detta förutsätts också gälla för indatafiler som läses in med `fromFile`.
Tips:

- Värdet `indexOfHeading` kan skapas med hjälp av metoden `zipWithIndex` som fungerar på alla sekvenssamlingar, samt metoden `toMap` som fungerar på sekvenser av 2-tupler. Undersök först hur metoderna fungerar i REPL och sök upp deras dokumentation.
- Skapa en indatafil som du kan använda för att testa att `Table` fungerar.

Specification **`Table`**

```
case class Table(
  data: Vector[Vector[String]],
```

```
  headings: Vector[String],
  sep: String){
  /** A 2-tuple with (number of rows, number of columns) in data */
  val dim: (Int, Int) = ???

  /** The element in row r an column c of data, counting from 0 */
  def apply(r: Int, c: Int): String = ???

  /** The row-vector r in data, counting from 0 */
  def row(r: Int): Vector[String]= ???

  /** The column-vector c in data, counting from 0 */
  def col(c: Int): Vector[String] = ???

  /** A map from heading to index counting from 0 */
  lazy val indexOfHeading: Map[String, Int] = ???

  /** The column-vector with heading h in data */
  def col(h: String): Vector[String] = ???

  /** A vector with the distinct, sorted values of col with heading h */
  def values(h: String): Vector[String] = ???

  /** Headings and data with columns separated by sep */
  override lazy val toString: String = ???
}
object Table {
  /** Creates a new Table from fileName with columns split by sep */
  def fromFile(fileName: String, separator: Char = ';'): Table = ???
}
```

b) Skapa med hjälp av `Table` ett program som kan köras från terminalen med `scala regtable infile.csv ';'` som ger en utskrift av antalet förekomster av olika värden i respektive kolumn (alltså en variant av registrering).

Uppgift 4. *Generiska funkioner.* En generisk funktion har (minst) en typparameter inom klammerparenteser efter namnet, till exempel `[T]`. Denna typ förekommer sedan som typ på (någon av) parametrarna i parameterlistan. Kompilatorn härleder en konkret typ vid kompileringstid och ersätter typparametern med denna konkreta typ. På så sätt kan en funktion fungera för många olika typer.

a) Förklara för varje rad nedan vad som händer.

```
1 scala> def tnirp[T](x: T): Unit = println(x.toString.reverse)
2 scala> tnirp(42)
3 scala> tnirp("hej")
4 scala> case class Gurka(vikt: Int)
5 scala> tnirp(Gurka(42))
6 scala> tnirp[String](42)
7 scala> tnirp[Double](42)
```

b) Man kan kombinera generiska funktioner med funktioner som tar funktioner som parametrar. Det är så `map` och `foreach` är implementerade. Förklara

för varje rad nedan vad som händer.

```
scala> def compose[A, B, C](f: A => B, g: B => C)(x: A): C = g(f(x))
scala> def inc(x: Int): Int = x + 1
scala> def half(x: Int): Double = x / 2.0
scala> compose(inc, half)(42)
scala> compose(half, inc)(42)
```

c) Hur lyder felmeddelandet på sista raden ovan? Ändra `inc` och/eller `half` så att typerna passar.

Uppgift 5. *Generiska klasser.* Även klasser kan vara generiska. En generisk klass har (minst) en typparameter inom klammerparenteser efter klassens namn.

a) Testa nedan generiska klass `Cell[T]` i REPL. Skapa instanser av klassen `Cell[T]` där typparametern `T` binds till olika konkreta typer och förklara vad som händer.

```
scala> class Cell[T](var value: T){
         override def toString = "Cell(" + value + ")"
       }
scala> new Cell(42)
scala> new Cell("hej")
scala> new Cell(new Cell(math.Pi))
scala> new Cell[String](42)
scala> new Cell[Double](42)
```

b) Lägg till metoden `def concat[U](that: Cell[U]):Cell[String]` i klassen `Cell` som konkatenerar strängrepresentationerna av de båda cellvärdena.

```
scala> val a = new Cell("hej")
scala> val b = new Cell(42)
scala> a concat b
```

c) Vilken sorts celler kan du konkatenera om du tar bort typparameternamnet `U` i `concat` samtidigt som du använder `Cell[T]` som typ på värdeparametern `that`? Vad ger det för konsekvenser för celler av annan typ än `Cell[String]`?

d) Denna uppgift illustrerar grunderna för att hur generiska samlingar är konstruerade, men vi går inte djupare här (det kommer mer i fördjupningskursen). Fundera om du vill på hur en generisk Matris-klass skulle kunna se ut och om du är intresserad av att fördjupa dig så gör fördjupningsuppgift 8.

Uppgift 6. *Matriser med array i Java.* Om man redan vid allokering vet hur många element en matris ska ha, använder man i Java gärna en array av arrayer. En heltalsmatris (en array av array av heltal) skrivs i Java med dubbla hakparentespar `int[][]` direkt efter typen. Vid allokering använder man nyckelordet `new` och antalet element i respektive dimension anges inom hakparenteserna; t.ex. så ger `new int[42][21]` en matris med 42 rader och 21 kolumner,

vilket motsvarar att man i Scala skriver[22] `Array.ofDim[Int](42,21)`. Alla element får defaultvärdet för typen, som är `0` för typen `Int` i Scala, motsvarande **`int`** i Java.

a) Skriv nedan program i en editor och spara koden i filen `ArrayMatrix.java` och kompilera med `javac ArrayMatrix.java` och kör i terminalen med `java ArrayMatrix` och undersök utskriften. Förklara vad som händer. Notera några skillnader i hur matriser används i Scala och Java.

```
// ArrayMatrix.java

public class ArrayMatrix {

    public static void showMatrix(int[][] m){
        System.out.println("\n--- showMatrix ---");
        for (int row = 0; row < m.length; row++){
            for (int col = 0; col < m[row].length; col++) {
                System.out.print("[" + row + "]");
                System.out.print("[" + col + "] = ");
                System.out.print(m[row][col] + "; ");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        System.out.println("ArrayMatrix test");
        int[][] xss = new int[10][5];
        showMatrix(xss);
    }
}
```

b) Implementera nedan metod `fillRnd` inuti klassen `ArrayMatrix`. Skriv kod som fyller matrisen `m` med slumptal mellan 1 och `n`.

```
    public static void fillRnd(int[][] m, int n){
        /* ??? */
    }
```

Tips: med detta uttryck skapas ett slumptal mellan 1 och 42 i Java:
`(int) (Math.random() * 42 + 1)`
där typkonverteringen `(int)` ger samma effekt som ett anrop av metoden `toInt` i Scala; alltså att dubbelprecisionsflyttal omvandlas till heltal genom avkortning av alla eventuella decimaler.

Ändra huvudprogrammet till:

[22] Ett annat längre, men kanske tydligare, sätt att skriva detta i Scala där initialvärdet framgår explicit: `Array.fill(42)(Array.fill(21)(0))`

```
    public static void main(String[] args) {
        System.out.println("ArrayMatrix test");
        int[][] xss = new int[10][5];
        showMatrix(xss);
        fillRnd(xss, 6);
        showMatrix(xss);
    }
```

Programmet ska ge en utskrift som liknar följande:

```
$ javac ArrayMatrix.java
$ java ArrayMatrix
ArrayMatrix test

--- showMatrix ---
[0][0] = 0; [0][1] = 0; [0][2] = 0; [0][3] = 0; [0][4] = 0;
[1][0] = 0; [1][1] = 0; [1][2] = 0; [1][3] = 0; [1][4] = 0;
[2][0] = 0; [2][1] = 0; [2][2] = 0; [2][3] = 0; [2][4] = 0;
[3][0] = 0; [3][1] = 0; [3][2] = 0; [3][3] = 0; [3][4] = 0;
[4][0] = 0; [4][1] = 0; [4][2] = 0; [4][3] = 0; [4][4] = 0;
[5][0] = 0; [5][1] = 0; [5][2] = 0; [5][3] = 0; [5][4] = 0;
[6][0] = 0; [6][1] = 0; [6][2] = 0; [6][3] = 0; [6][4] = 0;
[7][0] = 0; [7][1] = 0; [7][2] = 0; [7][3] = 0; [7][4] = 0;
[8][0] = 0; [8][1] = 0; [8][2] = 0; [8][3] = 0; [8][4] = 0;
[9][0] = 0; [9][1] = 0; [9][2] = 0; [9][3] = 0; [9][4] = 0;

--- showMatrix ---
[0][0] = 6; [0][1] = 2; [0][2] = 6; [0][3] = 3; [0][4] = 5;
[1][0] = 2; [1][1] = 4; [1][2] = 6; [1][3] = 1; [1][4] = 1;
[2][0] = 5; [2][1] = 4; [2][2] = 4; [2][3] = 1; [2][4] = 5;
[3][0] = 4; [3][1] = 6; [3][2] = 6; [3][3] = 1; [3][4] = 3;
[4][0] = 4; [4][1] = 6; [4][2] = 2; [4][3] = 3; [4][4] = 2;
[5][0] = 2; [5][1] = 4; [5][2] = 5; [5][3] = 5; [5][4] = 3;
[6][0] = 6; [6][1] = 5; [6][2] = 2; [6][3] = 4; [6][4] = 3;
[7][0] = 1; [7][1] = 6; [7][2] = 1; [7][3] = 6; [7][4] = 2;
[8][0] = 1; [8][1] = 1; [8][2] = 5; [8][3] = 3; [8][4] = 2;
[9][0] = 1; [9][1] = 1; [9][2] = 1; [9][3] = 5; [9][4] = 4;
```

9.2 Extrauppgifter

Uppgift 7. *Skapa ett yatzy-spel för användning i terminalen.*

a) Skapa med en editor en klass enligt nedan specifikation. Läs om hur de olika predikaten för att kolla olika giltiga kombinationer i Yatzy ska fungera här: en.wikipedia.org/wiki/Yahtzee. Bygg ett huvudprogram som testar dina funktioner. Kompilera och testa i terminalen allteftersom du lägger till nya funktioner.

Specification **YatzyRows**

```
/** En skiss på en klass som kan användas till ett förenklat yatzy-spel */
case class YatzyRows(val rows: Vector[Vector[Int]]) {
```

```
  /** A new YatzyRows with a new row of 5 dice rolls appended to rows  */
  def roll: YatzyRows = ???

  /** A new YatzyRows with some indices of the last row re-rolled  */
  def reroll(indices: Vector[Int]): YatzyRows = ???
}

object YatzyRows {
  def isYatzy(xs: Vector[Int]): Boolean = ???
  def isThreeOfAKind(xs: Vector[Int]): Boolean = ???
  def isFourOfAKind(xs: Vector[Int]): Boolean = ???
  def isFullHouse(xs: Vector[Int]): Boolean = ???
  def isSmallStraight(xs: Vector[Int]): Boolean = ???
  def isLargeStraight(xs: Vector[Int]): Boolean = ???
}
```

b) Använd `YatzyRows` för att med hjälp av många tärningskast beräkna sannolikheter för några olika giltiga kombinationer. Använd, om du vill, möjligheten som reglerna ger att slå om tärningar i två ytterliggare kast, där de tärningar som slås om väljs slumpmässigt.

c) Bygg ett förenklat yatzy-spel i terminalen där användaren kan bestämma vilka tärningar som ska slås om. Använd `Scanner` för att läsa indata från användaren. Börja med något riktigt enkelt och bygg sedan vidare på ditt spel genom att införa fler och fler funktioner.

9.3 Fördjupningsuppgifter

TODO!!!

Uppgift 8. *Skapa en generisk, oföränderlig matrisklass.* (TODO!!! Denna uppgift är inte färdig men tanken är att göra ngt kul med Matrix, kanske en SpriteEditor och ett enkelt SpriteGame där man har ett bräde med pjäser eller ngt) Med hjälp av en typparameter kan vi skapa en matrisklass som kan innehålla vilka element som helst. Implementera nedan specifikation. Testa din matrisklass i REPL för olika typer av element.

Specification **`Matrix[T]`**

```
case class Matrix[T](data: Vector[Vector[T]]){

  def map[U](f: T => U): Matrix[U] = Matrix(data.map(_.map(f)))

  def foreachRowCol[U](f: (Int, Int, T) => Unit): Unit =
    for (r <- 0 until data.size) {
      for (c <- 0 until data(r).size) {
        f(r, c, data(r)(c))
      }
    }

  /** The element at row r and column c */
  def apply(r: Int, c: Int): T = ???
```

```
  /** Gives Some[T](element) at row r and column c
   *  if r and c are within index bounds, else None */
  def get(r: Int, c: Int): Option[T] = ???

  /** The row vector of row r */
  def row(r: Int): Vector[T] = ???

  /** The column vector of column c */
  def col(c: Int): Vector[T] = ???

  /** A new Matrix with element at row r and col c updated */
  def updated(r: Int, c: Int, value: T): Matrix[T] = ???
}
object Matrix {
  def fill[T](rowSize: Int, colSize: Int)(init: T): Matrix[T] =
    new Matrix(Vector.fill(rowSize)(Vector.fill(colSize)(init)))
}
```

a) Använd ovan matrisklass för att göra en SpriteEditor med JColorChoser enligt nedan skiss.

```
object ColorChooser {
  import java.awt.Color
  import javax.swing.JColorChooser

  var title = "Pick Color"
  private val chooser = new JColorChooser(Color.BLACK)
  private val dialog = JColorChooser.
    createDialog(null, title, true, jcs, null, null)

  def getColor(initColor: Color = Color.BLACK): Color = {
    chooser.setColor(initColor)
    dialog.setVisible(true)
    chooser.getColor
  }
}

class Sprite(
  val id: String,
  val size: (Int, Int),
  val pixels: Matrix[Option[Int]], // None if transparent otherwise color num
  var scale: Int,
  var colors: Vector[java.awt.Color],
  var pos: (Int, Int, Int)  // (row, col, layer)
){
  def row = pos._1
  def col = pos._2
  def layer = pos._3
}
```

```
class SpriteEditor(rows: Int = 64, cols: Int = 64, scale: Int = 16, nColors: Int = 16)
  private val w = new SimpleWindow(???)
  def edit: Unit = ???
}
```

Uppgift 9. TODO!!! *Klasser för täta och glesa matematiska matriser med flyttal.* FUNDERA PÅ: om detta är för svårt när man inte läst linalg...

a) Skapa en oföränderlig, final klass `DenseMatrix` för matematiska matriser med dubbelprecisionsflyttal. `DenseMatrix` ska internt lagra elementen i en privat *endimensionell* array av flyttal av typen `Array[Double]`. Klassen ska inte vara en case-klass. Det ska gå att skapa matriser med uttrycket DenseMatrix.ofDim(3,7)(1.0,42,3.2,1.0,2.2,3) tack vare ett kompanjonsobjekt med lämplig fabriksmetod som anropar den privata konstruktorn. Om antalet element är för litet i förhållande till den angivna dimensionen så fyll på med nollor. Om det är för många tal i förhållande till dimensionen så kasta ett undantag TODO!!! tror jag??

b) Överskugga metoderna equals och hashcode och ge `DenseMatrix` innehållslikhet i stället för referenslikhet.

c) Implementera egna innehålllikhetsmetoder med namnet `===` på `DenseMatrix` som är typsäker, d.v.s. bara tillåter jämförelse mellan matriser.

d) *SparseMatrix med private mutable.Map[(Int, Int), Double]* som bara lagrar index som inte är noll.

e) Implementera addition, subtraktion och multiplikation av matriser.

f) Skapa ett trait Matrix som både DenseMatrix och SparseMatrix ärver, med lämpliga abstrakta och konkreta medlemmar.

Uppgift 10. *Matriser med `ArrayList` i Java.* Om man i Java inte vet antalet element i matrisen från början kan man använda en lista av typen `ArrayList`, där varje element i sin tur innehåller en lista av typen`ArrayList`. Javas `ArrayList` är en generisk samling som motsvaras av Scalas `ArrayBuffer`. Generiska samlingar i Java kan endast innehålla referenstyper; vill man ha en primitiv typ, t.ex. `int`, behöver man packa in denna i en s.k. wrapper-klass, t.ex. klassen `Integer`. Det finns en wrapper-klass för varje primitiv typ i Java. Matristypen för en heltalstyp i Java skrivs `ArrayList<ArrayList<Integer>>` där alltså `<T>` motsvarar Scalas hakparenteser `[T]` för typparametern T.

a) TODO!!! Hitta på deluppgifter med `ArrayList<ArrayList<Integer>>` som illustrerar ovan. Peka framåt till scalajava-veckan.

10. Övning: sorting

Mål

- ☐ Förstå hur sorteringsordningen är definierad för strängar.
- ☐ Förstå skillnaderna mellan strängjämförelser i Scala och Java, samt kunna jämföra strängar med jämförelsoperatorer i Scala och med `compareTo` i Java.
- ☐ Kunna sortera sekvenssamlingar innehållande objekt av grundtyper med hjälp av inbyggda och egendefinierade sorteringsordningar med metoderna `sorted`, `sortBy` och `sortWidth`.
- ☐ Kunna använda inbyggda linjärsöknings- och binärsökningsmetoder.
- ☐ Kunna implementera en egen sökalgoritm med linjärsökning och binärsökning.
- ☐ Förstå när binärsökning är lämplig och möjlig.
- ☐ Kunna implementera en enkel sorteringsalgoritm, t.ex. insättningssortering eller urvalssortering, både till ny samling och på plats.
- ☐ Känna till hur implicita sorteringsordningar används för grundtyperna och egendefinierade typer.
- ☐ Känna till existensen av, funktionen hos, och relationen mellan klasserna `Ordering` och `Comparator`, samt `Ordered` och `Comparable`.

Förberedelser

- ☐ Studera begreppen i kapitel **??**.

10.1 Grunduppgifter

Uppgift 1. *Jämföra strängar i Scala.* I Scala kan strängar jämföras med operatorerna ==, !=, <, <=, >, >=, där likhet/olikhet avgörs av om alla tecken i strängen är lika eller inte, medan större/mindre avgörs av sorteringsordningen i enlighet med varje teckens Unicode[23]-värde.

a) Vad ger följande jämförelser för värde?

```
scala> 'a' < 'b'
scala> "aaa" < "aaaa"
scala> "aaa" < "bbb"
scala> "AAA" < "aaa"
scala> "ÄÄÄ" < "ÖÖÖ"
scala> "ÅÅÅ" < "ÄÄÄ"
```

Tyvärr så följer ordningen av ÄÅÖ inte svenska regler, men det ignorerar vi i fortsättningen för enkelhets skull; om du är intresserad av hur man kan fixa detta, gör uppgift 19.

✎ b) Vilken av strängarna $s1$ och $s2$ kommer först (d.v.s. är ”mindre”) om $s1$ utgör början av $s2$ och $s2$ innehåller fler tecken än $s1$?

[23] sv.wikipedia.org/wiki/Unicode

Uppgift 2. *Jämföra strängar i Java.* I Java kan man **inte** jämföra strängar med operatorerna <, <=, >, och >=. Dessutom ger operatorerna == och != inte innehålls(o)likhet utan referens(o)likhet. Istället får man använda metoderna `equals` och `compareTo`, vilka också fungerar i Scala eftersom strängar i Scala och Java är av samma typ, nämligen `java.lang.String`.

a) Vad ger följande uttryck för värde?

```
scala> "hej".getClass.getTypeName
scala> "hej".equals("hej")
scala> "hej".compareTo("hej")
```

b) Studera dokumentationen för metoden `compareTo` i `java.lang.String`[24] och skriv minst 3 olika uttryck i Scala REPL som testar hur metoden fungerar i olika fall.

c) Studera dokumentationen `compareToIgnoreCase` [25] och skriv minst 3 olika stränguttryck i Scala REPL som testar hur metoden fungerar i olika fall.

d) Vad skriver följande Java-program ut?

```
public class StringEqTest {
    public static void main(String[] args){
        boolean eqTest1 =
          (new String("hej")) == (new String("hej"));
        boolean eqTest2 =
          (new String("hej")).equals(new String("hej"));
        int eqTest3 =
          (new String("hej")).compareTo(new String("hej"));
        System.out.println(eqTest1);
        System.out.println(eqTest2);
        System.out.println(eqTest3);
    }
}
```

Uppgift 3. *Sortering med inbyggda sorteringsmetoder.* För grundtyperna (`Int`, `Double`, `String`, etc.) finns en fördefinierad ordning som gör så att färdiga sorteringsmetoder fungerar på alla samlingar i `scala.collection`. Även jämförelseoperatorerna i uppgift 1 fungerar enligt den fördefinierade ordningsdefinitionen för alla grundtyper. Denna ordningsdefinition är *implicit tillgänglig* vilket betyder att kompilatorn hittar ordningsdefinitionen utan att vi explicit måste ange den. Detta fungerar i Scala även med primitiva `Array`.

a) Testa metoden `sorted` på några olika samlingar. Förklara vad som händer. Hur lyder felmeddelande på sista raden? Varför blir det fel?

```
scala> Vector(1.1, 4.2, 2.4, 42.0, 9.9).sorted
scala> val xs = (100000 to 1 by -1).toArray
```

[24] docs.oracle.com/javase/8/docs/api/java/lang/String.html#compareTo-java.lang.String-

[25] docs.oracle.com/javase/8/docs/api/java/lang/String.html#compareToIgnoreCase-java.lang.String-

```
scala> xs.sorted
scala> xs.map(_.toString).sorted
scala> xs.map(_.toByte).sorted.distinct
scala> case class Person(firstName: String, familyName: String)
scala> val ps = Vector(Person("Zeb", "Robinson"), Person("Robin","Zebson"))
scala> ps.sorted
```

b) Om man har en samling med egendefinierade klasser eller man vill ha en annan sorteringsordning får man definiera ordningen själv. Ett helt generellt sätt att göra detta på illustreras i uppgift 16, men de båda hjälpmetoderna `sortWith` och `sortBy` räcker i de flesta fall. Hur de används illustreras nedan. Metoden `sortBy` kan användas om man tar fram ett värde av grundtyp och är nöjd med den inbyggda sorteringsordningen.

Metoden `sortWith` används om man vill skicka med ett eget jämförelsepredikat som ordnar två element; funktionen ska returnera **`true`** om det första elementet ska vara först, annars **`false`**.

```
scala> case class Person(firstName: String, familyName: String)
scala> val ps = Vector(Person("Zeb", "Robinson"), Person("Robin","Zebson"))
scala> ps.sortBy(_.firstName)
scala> ps.sortBy(_.familyName)
scala> ps.sortBy  // tryck TAB två gånger för att se signaturen
scala> ps.sortWith((p1, p2) => p1.firstName > p2.firstName)
scala> ps.sortWith  // tryck TAB två gånger för att se signaturen
scala> Vector(9,5,2,6,9).sortWith((x1, x2) => x1 % 2 > x2 % 2)
```

Vad har metoderna `sortWith` och `sortBy` för signaturer?

c) Lägg till attributet `age: Int` i case-klassen `Person` ovan och lägg till fler personer med olika namn och ålder i en vektor och sortera den med `sortBy` och `sortWith` för olika attribut. Välj själv några olika sätt att sortera på.

Uppgift 4. *Tidmätning.* I kommande uppgifter kommer du att ha nytta av funktionen `timed` enligt nedan.

```
def timed[T](code: => T): (T, Long) = {
  val now = System.nanoTime
  val result = code
  val elapsed = System.nanoTime - now
  println(s"\ntime: ${elapsed / 1e6}$ ms")
  (result, elapsed)
}
```

a) Klistra in `timed` i REPL och testa så att den fungerar, genom att mäta hur lång tid nedan uttryck tar att exekvera.

```
scala> val (v, t1) = timed{ (1 to 1000000).toVector.reverse }
scala> val (s, t2) = timed{ v.toSet }
scala> timed{ v.find(_ == 1) }
scala> timed{ s.find(_ == 1) }
scala> timed{ s.contains(1) }
```

b) Försök förklara skillnaderna i exekveringstid mellan de olika sätten att söka reda på talet 1 i samlingen. Ungefär hur många gånger behöver man använda `contains` på heltalsmängden `s` för att det ska löna sig att skapa `s` i stället för att linjärsöka i `v` med `find` i ovan exempel?

Uppgift 5. *Sökning med inbyggda sökmetoder.*

a) *Linjärsökning framifrån med `indexOfSlice`.* Studera dokumentationen för Scalas samlingsmetod `indexOfSlice`[26] och skriv 8 olika uttryck i REPL som, både med en sträng och med en vektor med heltal, provar 4 olika fall: (1) finns i börja, (2) finns någonstans i mitten, (3) finns i slutet, samt (4) finns ej.

b) *Linjärsökning bakifrån med `lastIndexOfSlice`.* Studera dokumentationen för Scalas samlingsmetod `lastIndexOfSlice`[27] och skriv 8 olika uttryck i REPL som, både med en sträng och med en vektor med heltal, provar 4 olika fall: (1) finns i börja, (2) finns någonstans i mitten, (3) finns i slutet, samt (4) finns ej.

c) *Sökning med inbyggd binärsökning.* Om en samling är sorterad kan man utnyttja detta för att göra snabbare sökning. Vid **binärsökning** (eng. *binary search*)[28] börjar man på mitten och kollar vilken halva att söka vidare i; sedan delar man upp denna halva på mitten och kollar vilken fjärdedel att söka vidare i, etc.

I objektet `scala.collection.Searching`[29] finns en metod `search` som, om den importeras, erbjuder binärsökning för alla sorterade sekvenssamlingar. Om samlingen är sorterad ger den ett objekt av case-klassen `Found` som innehåller indexet för platsen där elementet först hittats; alternativt om det som eftersöks ej finns, ges ett objekt av case-klassen `InsertionPoint` som innehåller indexet där elementet borde ha varit placerad om det funnits i samlingen. Observera att om samlingen inte är sorterad är resultatet ”odefinierat”, d.v.s. något returneras men det är *inte* att lita på; man måste alltså först sortera samlingen eller vara helt säker på att den är sorterad.

Undersök hur `search` fungerar genom att förklara vad som händer nedan. Vilken är snabbast av `lin` och `bin` nedan? Använd `timed` från uppgift 4.

```
scala> val udda = (1 to 1000000 by 2).toVector
scala> import scala.collection.Searching._
scala> udda.search(udda.last)
scala> udda.search(udda.last + 1)
scala> udda.reverse.search(udda(0))
scala> def lin(x: Int, xs: Seq[Int]) = xs.indexOf(x)
scala> def bin(x: Int, xs: Seq[Int]) = xs.search(x) match {
         case Found(i) => i
         case InsertionPoint(i) => -i
       }
scala> timed{ lin(udda.last, udda) }
scala> timed{ bin(udda.last, udda) }
```

[26] docs.scala-lang.org/overviews/collections/seqs.html
[27] docs.scala-lang.org/overviews/collections/seqs.html
[28] en.wikipedia.org/wiki/Binary_search_algorithm
[29] http://www.scala-lang.org/api/current/#scala.collection.Searching$

d) Om en samling innehåller n element, hur många jämförelser behövs då vid binärsökning i värsta fall? *Tips:* Läs om algoritmen på wikipedia[28].

Uppgift 6. *Sök bland LTH:s kurser med linjärsökning*

a) Surfa till denna URL:
`http://kurser.lth.se/lot/?lasar=16_17&soek_text=&sort=kod&val=kurs&soek=t`
och inspektera html-koden i din webbläsare genom att trycka *Ctrl+U* (fungerar i Firefox och Chrome). Rulla ner till rad 171 och framåt. Var finns antalet poäng för resp kurs i html-koden?

b) Klistra in objektet `courses` med kommandot `:paste` i REPL.[30] Vad gör koden? Hur många kurser innehåller `lth2016`?

```
object courses {
  def download(year: String = "16_17"): Vector[Course]  = {
    val urlStart = s"http://kurser.lth.se/lot/?lasar=$year"
    val urlSearch = "&soek_text=&sort=kod&val=kurs&soek=t"
    val url = urlStart + urlSearch
    println("*** Downloading from: " + url)
    println("*** This may take a while...")
    val lines = scala.io.Source.fromURL(url).getLines.toVector
    lines.filter(_.contains("kurskod")).map(Course.fromHtml)
  }

  lazy val lth2016: Vector[Course] = download()

  case class Course(
    code:    String,
    nameSv:  String,
    nameEn:  String,
    credits: Double,
    level:   String
  )

  object Course {
    import scala.util.Try
    def fromHtml(s: String): Course = {
      def extract(s: String, init: String, stop: Char): String =
        s.replaceAllLiterally(init, "").takeWhile(_ != stop)
      val codeInit = """<a href="/lot/?val=kurs&amp;kurskod="""
      val dataInit = """<td class="mitt">"""
      val xs = s.split("td>")
      val code = Try { extract(xs(1), codeInit, '"') }.getOrElse("???")
      val credits = Try {
        val s = extract(xs(2), dataInit, '<')
        s.replaceAllLiterally(",",".").toDouble //fix decimals
      }.getOrElse(0.0)
      val level = Try { extract(xs(3), dataInit, '<') }.getOrElse("???")
      val nameSv = Try { xs(5).takeWhile(_ != '<') }.getOrElse("???")
      val nameEn = Try { xs(7).takeWhile(_ != '<') }.getOrElse("???")
      Course(code, nameSv, nameEn, credits, level)
```

[30] Du kan ladda ner koden från:
github.com/lunduniversity/introprog/tree/master/compendium/examples/lth-courses/courses.scala

```
    }
  }
}
```

c) *Linjärsökning med find.* Teknologen Oddput Clementina vill gå första bästa datavetenskapskurs som är på G2-nivå. Hjälp Oddput med att söka upp första bästa kurs genom linjärsökning med samlingsmetoden `find`. Kurskoder vid datavetenskap börjar på EDA eller ETS[31]. *Tips:* Du har nytta av att definiera predikatet `def isCS(s: String): Boolean` som i sin tur lämpligen nyttjar strängmetoden `startsWith`.

d) *Implementera linjärsökning.* Som träning ska du nu implementera en egen linjärsökningsfunktion med signaturen:
`def linearSearch[T](xs: Seq[T])(p: T => Boolean): Int = ???`
Funktionen ska ta en sekvenssamling `xs` och ett predikat `p` som är en funktion som tar ett element och returnerar ett booleskt värde. Funktionen `p` ska ge true om parametern är ett eftersökt element. Funktionen `linearSearch` ska returnera index för första hittade elementet i `xs` där `p` gäller. Om det inte finns något element som uppfyller predikatet ska -1 returneras. Skriv först pseudokod för funktionen med penna och papper. Använd `while`.

Typen `Seq` är supertyp till alla sekvenssamlingar, så om vi använder den som parametertyp för parametern `xs` så fungerar funktionen för `Vector`, `Array`, `List`, etc. Genom typparametern `T` blir funktionen generisk och fungerar för godtycklig typ.

e) Skriv i en editor en funktion `def rndCode: String` som genererar slumpmässiga kurskoder som består av 6 tecken enligt dessa regler: de första tre tecknen är bokstäver mellan A och Z, de sista två är siffror mellan 0 och 9, medan det fjärde tecknet kan vara antingen en siffra mellan 0 och 9 eller ett av dessa tecken: `ACFGLMNP`. *Tips:* Använd REPL för att stegvis bygga upp hjälpfunktioner som du, när de fungerar som de ska, klistrar in i ett editorfönster som lokala funktioner där du utvecklar den slutliga koden för en lättläst, concis och fungerande `rndCode`.

f) Använd `rndCode` från föregående deluppgift för att fylla en vektor kallad `xs` med en halv miljon slumpmässiga kurskoder. För varje slumpkod i `xs` sök med din funktion `linearSearch` efter index i vektorn `courses.lth2016` från deluppgift b. Mät totala tiden för de 500000 linjärsökningarna med hjälp av funktionen `timed` från uppgift 4. Hur många av de slumpmässiga kurskoderna hittades bland de verkliga kurskoderna på LTH?

g) Hur kan du implementera `linearSearch` med den inbyggda samlingsmetoden `indexWhere`? 

[31] Detta är en förenklad bild av LTH:s kurskodnamnsystem. Några kurser från EIT-institutionen kommer att slinka med, men det bortser vi ifrån i denna uppgift.

Uppgift 7. *Sök bland LTH:s kurser med binärsökning.*
Sökningsalgoritmen BINSEARCH kan formuleras med nedan pseudokod:

Indata : En växande sorterad sekvens xs med n heltal och ett eftersökt heltal key
Resultat : Ett heltal $i \geq 0$ som anger platsen där x finns, eller ett negativt tal i där $-i$ motsvarar platsen där x ska sättas in i sorterad ordning om x ej finns i samlingen.

```
1  sätt intervallet (low, high) till (0, n − 1)
2  found ← false
3  mid ← −1
4  while low ≤ high and not found do
5  |  mid ← platsen mitt emellan low och high
6  |  if xs(mid) == key then
7  |  |  found ← true
8  |  else
9  |  |  if xs(mid) < key then
10 |  |  |  low ← mid + 1
11 |  |  else
12 |  |  |  high ← mid − 1
13 |  |  end
14 |  end
15 end
16 if found then
17 |  return mid
18 else
19 |  return −(low + 1)
20 end
```

a) Prova algoritmen ovan med penna och papper på en sorterade sekvens med mindre än 10 heltal. Prova om algoritmen fungerar med ett jämt antal tal, ett udda antal tal, en sekvens med ett heltal och en tom sekvens. Prova både om talet du letar efter finns och om det inte finns.

b) Implementera binärsökning i en funktion med signaturen
`def binarySearch(xs: Seq[String], key: String): Int = ???`
och testa i REPL för olika fall. Vad händer om sekvensen inte är sorterad?

c) Använd `binarySearch` för att leta efter LTH-kurser enligt nedan. Använd `rndCode`, `timed` och `courses` från tidigare uppgifter.

```
def binarySearch(xs: Seq[String], key: String): Int = ???

val lthCodesSorted = courses.lth2016.map(_.code).sorted
val xs = Vector.fill(500000)(rndCode)
val (_, elapsedBin) =
  timed{xs.map(x => binarySearch(lthCodesSorted, x))}
val (_, elapsedLin) =
  timed{xs.map(x => linearSearch(lthCodesSorted)(_ == x))}
```

```
println(elapsedLin / elapsedBin)
```

d) Hur mycket snabbare blev binärsökningen jämfört med linjärsökningen?[32]

Uppgift 8. *Linjärsökning i Java.* Denna uppgift bygger vidare på uppgift 6 i kapitel **??**. Du ska göra en variant på linjärsökning som innebär att leta upp första yatzy-raden i en matris där varje rad innehåller utfallet av 5 tärningskast.

a) Du ska lägga till metoderna `isYatzy` och `findFirstYatzyRow` i klassen `ArrayMatrix` i uppgift 6 i kapitel **??** enligt nedan skiss. Vi börjar med metoden `isYatzy` i denna deluppgift (nästa deluppgift handlar om `findFirstYatzyRow`). OBS! Det finns en bug i `isYatzy` – rätta bugen och testa så att den fungerar.

```
    public static boolean isYatzy(int[] dice){ /* has one bug! */
        int col = 1;
        boolean allSimilar = true;
        while (col < dice.length && allSimilar) {
          allSimilar = dice[0] == dice[col];
        }
        return allSimilar;
    }

    /** Finds first yatzy row in m; returns -1 if not found */
    public static int findFirstYatzyRow(int[][] m){
        int row = 0;
        int result = -1;
        while (???) {
             /* linear search  */
        }
        return result;
    }
```

b) Implementera `findFirstYatzyRow`. Skapa först pseudo-kod för länjärsökningsalgoritmen innan du skriver implementationen i Java. Testa ditt program genom att lägga till följande rader i huvudprogrammet. Metoden `fillRnd` ingår i uppgift 6 i kapitel **??**.

```
        int[][] yss = new int[2500][5];
        fillRnd(yss, 6);
        int i = findFirstYatzyRow(yss);
        System.out.println("First Yatzy Index: " + i);
```

Uppgift 9. Implementera sortering av en heltalssekvens till en *ny* sekvens med **insättningssortering** (eng. *insertion sort*) i en funktion med följande signatur:

[32] Vid en körning på en i7-4970K med 4.0GHz tog `elapsedLin` cirka 3000 *ms* och `elapsedBin` cirka 60 *ms*. Binärsökning var alltså i detta fall ungefär 50 gånger snabbare än linjärsökning.

```
def insertionSort(xs: Seq[Int]): Seq[Int] = ???
```

Lösningsidé: Skapa en ny, tom sekvens som ska bli vårt sorterade resultat. För varje element i den osorterade sekvensen: Sätt in det på rätt plats i den nya sorterade sekvensen.

a) *Pseduokod:* Kör nedan pseudokod med papper och penna t.ex. på sekvensen 5 1 4 3 2 1. Rita minnessituationen efter varje runda i loopen. Här använder vi internt i funktionen föränderliga `ArrayBuffer` som är snabb på insättning och avslutar med `toVector` så att vi lämnar ifrån oss en oföränderlig sekvens.

```
1 result ← en ny, tom ArrayBuffer
2 foreach element e in xs do
3 |   pos ← leta upp rätt position i result
4 |   stoppa in e på plats pos i result
5 end
6 result.toVector
```

b) Implementera `insertionSort`. Använd en `while`-loop för att implementera rad 3 i pseduokoden. Sök upp dokumentationen för metoden `insert` på `ArrayBuffer`. Testa `insert` på `ArrayBuffer` i REPL och verifiera att den kan användas för att stoppa in på slutet på den ”oanvända” positionen som är precis efter sista positionen. Vad händer om man gör `insert` på positionen `size + 2`?

Klistra in din implementation av `insertionSort` i REPL och testa så att allt fungerar:

```
scala> insertionSort(Vector())
res0: Seq[Int] = Vector()

scala> insertionSort(Vector(42))
res1: Seq[Int] = Vector(42)

scala> insertionSort(Vector(1,2,3))
res2: Seq[Int] = Vector(1, 2, 3)

scala> insertionSort(Vector(5,1,4,3,2,1))
res3: Seq[Int] = Vector(1, 1, 2, 3, 4, 5)
```

Uppgift 10. Implementera sortering på plats (eng. *in-place*) i en `Array[String]` med urvalssortering (eng. *selection sort*)

Lösningsidé: För alla index i: sök $minIndex$ för ”minsta” strängen från plats i till sista plats och byt plats mellan strängarna på plats i och plats $minIndex$. Se även animering här: sv.wikipedia.org/wiki/Urvalssortering

Implementera enligt nedan skiss. *Tips:* Du har nytta av en modifierad variant av lösningen till uppgift 17 i kapitel **??**.

```
def selectionSortInPlace(xs: Array[String]): Unit = {
  def indexOfMin(startFrom: Int): Int = ???
  def swapIndex(i1: Int, i2: Int): Unit = ???
```

```
  for (i <- 0 to xs.size - 1) swapIndex(i, indexOfMin(i))
}
```

10.2 Extrauppgifter

Uppgift 11. *Undersök om en sekvens är sorterad.* Ett enkelt och lättläst sätt att undersöka om en sekvens är sorterad visas nedan.

```
1 scala> def isSorted(xs: Vector[Int]): Boolean = xs == xs.sorted
```

a) Om `xs` har 10^6 element, hur många jämförelser kommer i värsta fall att ske med `isSorted` enligt ovan. Metoden `sorted` använder algoritmen Timsort[33]. Sök upp antalet jämförelser i värstafallet på wikipedia.======= Denna lösning är dock relativt långsam för stora samlingar. Man behöver ju inte först sortera för att avgöra om det är sorterat (om man inte ändå hade tänkt sortera av andra skäl), det räcker att kolla att elementen är i växande ordning.

b) Om `xs` har n element, ungefär hur många jämförelser kommer i värsta fall att ske med `isStorted` ovan om man alltså först ska sortera och sedan jämföra den osorterade och den sorterade samlingen element för element? Metoden `sorted` använder algoritmen Timsort[34]. Sök upp värstafallsprestandan för Timsort på wikipedia. [35]

c) Implementera en effektivare variant av `isSorted` som använder en **`while`**-sats och kollar att elementen är i växande ordning.

d) Vad blir antalet jämförelser i värstafallet med metoden i deluppgift c om du har n element?

e) Man kan kolla om en sekvens är sorterad med det listiga tricket att först zippa sekvensen med sin egen svans och sedan kolla om alla element-par uppfyller sorteringskriteriet, alltså `xs.zip(xs.tail).forall(???)` där ??? byts ut mot lämpligt predikat. Vilken typ har 2-tupeln `xs.zip(xs.tail))` om `xs` är av typen `Vector[Int]`? Implementera `isSorted` med detta listiga trick. (Senare, i fördjupningsuppgift 15, ska vi göra `isSorted` generellt användbar för olika typer och olika ordningsdefinitioner.) ======= f) Man kan kolla om en sekvens är sorterad med det listiga tricket att först zippa sekvensen med sin egen svans och sedan kolla om alla element-par uppfyller sorteringskriteriet, alltså `xs.zip(xs.tail).forall(???)` där ??? byts ut mot lämpligt predikat. Vilken typ har 2-tupeln `xs.zip(xs.tail))` om `xs` är av typen `Vector[Int]`? Implementera `isSorted` med detta listiga trick. (I fördjupningsuppgift 15 görs denna variant av `isSorted` generellt användbar för olika typer och olika ordningsdefinitioner.)

[33] stackoverflow.com/questions/14146990/what-algorithm-is-used-by-the-scala-library-method-vector-sorted

[34] stackoverflow.com/questions/14146990/what-algorithm-is-used-by-the-scala-library-method-vector-sorted

[35] en.wikipedia.org/wiki/Timsort

Uppgift 12. Implementera och testa sortering på plats i en array med heltal med *instickssortering*[36].

a) Implementera och testa funktionen nedan i Scala med följande signatur:

```
  def insertionSort(xs: Array[Int]): Unit
```

Placera metoden i ett objekt med lämpligt namn, samt skapa ett huvudprogram med testkod. Kompilera och kör från terminalen. Börja med att skriva sorteringsalgoritmen i pseudokod.

b) Implementera och testa metoden nedan i Java med följande signatur:

```
  public static void insertionSort(int[] xs)
```

Placera metoden i en klass med lämpligt namn, samt skapa ett huvudprogram med testkod. Börja med att skriva sorteringsalgoritmen i pseudokod.

Uppgift 13. Implementera och testa sortering till ny sekvens med urvalssortering[37] i Scala, enligt nedan skiss. *Tips:* Du har nytta av lösningen till uppgift 17 i kapitel **??**.

```
def selectionSort(xs: Seq[String]): Seq[String] = {
  def indexOfMin(xs: Seq[String]): Int = ???
  val unsorted = xs.toBuffer
  val result = scala.collection.mutable.ArrayBuffer.empty[String]
  /*
  så länge unsorted inte är tom {
    minPos = indexOfMin(unsorted)
    elem   = unsorted.remove(minPos)
    result.append(elem)
  }
  */
  result.toVector
}
```

10.3 Fördjupningsuppgifter

Uppgift 14. *Typklasser och implicita parametrar.* I Scala finns möjligheter till avancerad funktionsprogrammering med s.k. **typklasser**, som definierar generella beteenden som fungerar för befintliga typer utan att implementationen av dessa befintliga typer behöver ändras. Vi nosar i denna uppgift på hur implicita argument kan användas för att skapa typklasser, illustrerat med hjälp av implicita ordningarna, som är en typisk och användbar tillämpning av konceptet typklasser.

[36] en.wikipedia.org/wiki/Insertion_sort
[37] en.wikipedia.org/wiki/Selection_sort

a) *Implicit parameter och implicit värde.* Med nyckelordet **`implicit`** framför en parameter öppnar man för möjligheten att låta kompilatorn ge argumentet ”automatiskt” om den kan hitta ett värde med passande typ som också är deklarerat med **`implicit`**, så som visas nedan.

```
scala> def add(x: Int)(implicit y: Int) = x + y
scala> add(1)(2)
scala> add(1)
scala> implicit val ngtNamn = 42
scala> add(1)
```

Vad blir felmeddelandet på rad 3 ovan? Varför fungerar det på rad 5 utan fel?

b) *Typklasser.* Genom att kombinera koncepten implicita värden, generiska klasser och implicita parametrar får man möjligheten att göra typklasser, så som `CanCompare` nedan, som vi kan få att fungera för befintliga typer utan att de behöver ändras.

Vad händer nedan? Vilka rader ger felmeddelande? Varför?

```
scala> trait CanCompare[T] { def compare(a: T, b: T): Int }
scala> def sort2[T](a: T, b: T)(implicit cc: CanCompare[T]): (T, T) =
         if (cc.compare(a, b) > 0) (b, a) else (a, b)
scala> sort2(42, 41)
scala> implicit object intComparator extends CanCompare[Int]{
         override def compare(a: Int, b: Int): Int = a - b
       }
scala> sort2(42, 41)
scala> sort2(42.0, 41.0)
```

c) Definiera ett implicit objekt som gör så att `sort2` fungerar för värden av typen `Double`.

d) Definiera ett implicit objekt som gör så att `sort2` fungerar för värden av typen `String`.

Uppgift 15. *Användning av implicit ordning.* Vi ska nu göra `isSorted` från uppgift 11 mer generellt användbar genom att möjliggöra att implicita ordningsfunktioner finns tillgängliga för olika typer.

a) Med signaturen `isSorted(xs: Vector[Int]): Boolean` så fungerar sorteringstestet bara för samlingar av typen `Vector[Int]`. Om vi i stället använder `isSorted(xs: Seq[Int]): Boolean` fungerar den för alla samlingar med heltal, även `Array` och `List`. Testa nedan funktion i REPL med heltalssekvenser av olika typ.

```
def isSorted(xs: Seq[Int]): Boolean = xs == xs.sorted
```

b) Men vi vill gärna att `isSorted` ska fungera för godtyckliga typer T som har en ordningsdefinition. Detta kan göras med nedan funktion eftersom metoden `sorted` är definierad för alla samlingar där typen `T` har en implicit ordning. Speciellt gäller detta för alla de grundtyperna `Int`, `Double`, `String`, etc.

```
def isSorted[T](xs: Seq[T]): Boolean = xs == xs.sorted
```

Testa metoden ovan i REPL enligt nedan.

```
scala> isSorted(Vector(1,2,3))
scala> isSorted(Array(1,2,3,1))
scala> isSorted(Vector("A","B","C"))
scala> isSorted(List("A","B","C","A"))
scala> case class Person(firstName: String, familyName: String)
scala> val persons = Vector(Person("Zeb", "Robson"), Person("Robin","Zebson"))
scala> isSorted(persons)
```

Vad ger sista raden för felmeddelande? Varför?

c) Vi vill gärna kunna jämföra element av godtycklig typ `T`, så att vi till exempel ska kunna implementera en generisk `isSorted` med **`while`** eller vårt `zip`-trick från uppgift 11f. Men det blir problem enligt nedan. Hur lyder felmeddelandet? Vad saknas?

```
scala> def isSorted[T](xs: Seq[T]): Boolean =
         xs.zip(xs.tail).forall(x => x._1 <= x._2)
```

d) Det blir även problem med denna implementation. Hur lyder felmeddelandet? Vad saknas?

```
scala> def isSorted[T](xs: Seq[T]): Boolean = xs == xs.sorted
```

e) *Implicita ordningar.* Man kan berätta för kompilatorn att den ska leta efter implicita ordningar av typen `T`. Detta kan göras genom att utöka signaturen för `isSorted` med en andra parameterlista, som tar en implicit parameter enligt följande:

```
def isSorted[T](xs: Seq[T])(implicit ord: Ordering[T]): Boolean =
  xs.zip(xs.tail).forall(x => ord.lteq(x._1, x._2))
```

Det finns fördefinierade implicita objekt `Ordering[T]` för alla grundtyper, alltså t.ex. `Ordering[Int]`, `Ordering[String]`, etc. Objekt av typen `Ordering` har jämförelsemetoder som t.ex. `lteq` (förk. *less than or equal*) och `gt` (förk. *greater than*). Testa så att kompilatorn hittar ordningen för samlingar med värden av några grundtyper. Kontrollera även enligt nedan att det fortfarande blir problem för egendefinierade klasser, t.ex. `Person` enligt tidigare (detta ska vi råda bot på i uppgift 16).

```
scala> isSorted(Vector(1,2,3))
scala> isSorted(Array(1,2,3,1))
scala> isSorted(Vector("A","B","C"))
scala> isSorted(List("A","B","C","A"))
scala> class Person(firsName: String, familyName: String)
scala> val persons = Vector(Person("Zeb", "Robson"), Person("Robin","Zebson"))
scala> isSorted(persons)
```

f) *Importera implicita ordningsoperatorer från en `Ordering`.* Om man gör import på ett `Ordering`-objekt får man tillgång till implicita konverteringar som gör att jämförelseoperatorerna fungerar. Testa nedan variant av `isSorted` på olika sekvenstyper och verifiera att <=, >, etc., nu fungerar enligt nedan.

```
def isSorted[T](xs: Seq[T])(implicit ord: Ordering[T]): Boolean = {
  import ord._
  xs.zip(xs.tail).forall(x => x._1 <= x._2)
}
```

Uppgift 16. *Skapa egen implicit ordning med* `Ordering`.

a) Ett sätt att skapa en egen, specialanpassad ordning är att mappa dina objekt till typer som redan har en implicit ordning. Med hjälp av metoden by i objektet `scala.math.Ordering` kan man skapa ordningar genom bifoga en funktion `T => S` där `T` är typen för de objekt du vill ordna och `S` är någon annan typ, t.ex. `String` eller `Int`, där det redan finns en implicit ordning.

```
scala> case class Team(name: String, rank: Int)
scala> val xs =
         Vector(Team("fnatic", 1499), Team("nip", 1473), Team("lumi", 1601))
scala> xs.sorted  // Hur lyder felmeddelandet? Varför blir det fel?
scala> val teamNameOrdering = Ordering.by((t: Team) => t.name)
scala> xs.sorted(teamNameOrdering)   //explicit ordning
scala> implicit val teamRankOrdering = Ordering.by((t: Team) => t.rank)
scala> xs.sorted   // Varför funkar det nu?
```

b) Vill man sortera i omvänd ordning kan man använda `Ordering.fromLessThan` som tar en funktion `(T, T) => Boolean` vilken ska ge **`true`** om första parametern ska komma före, annars **`false`**. Om vi vill sortera efter `rank` i omvänd ordning kan vi göra så här:

```
scala> val highestRankFirst =
         Ordering.fromLessThan[Team]((t1, t2) => t1.rank > t2.rank)
scala> xs.sorted(highestRankFirst)
```

c) Om du har en case-klass med flera fält och vill ha en fördefinierad implicit sorteringsordning samt även erbjuda en alternativ sorteringsordning kan du placera olika ordningsdefinitioner i ett kompanjonsobjekt; detta är nämligen ett av de ställen där kompilatorn söker efter eventuella implicita värden innan den ger upp att leta.

```
case class  Team(name: String, rank: Int)
object Team {
  implicit val highestRankFirst = Ordering.fromLessThan[Team]{
    (t1, t2) => t1.rank > t2.rank
  }
  val nameOrdering = Ordering.by((t: Team) => t.name)
}
```

```
scala> :pa
// Exiting paste mode, now interpreting.
case class  Team(name: String, rank: Int)
object Team {
  implicit val highestRankFirst =
```

```
    Ordering.fromLessThan[Team]{(t1, t2) => t1.rank > t2.rank}
  val nameOrdering = Ordering.by((t: Team) => t.name)
}
scala> val xs =
        Vector(Team("fnatic", 1499), Team("nip", 1473), Team("lumi", 1601))
scala> xs.sorted
scala> xs.sorted(Team.nameOrdering)
```

d) Det går också med kompanjonsobjektet ovan att få jämförelseoperatorer att fungera med din case-klass, genom att importera medlemmarna i lämpligt ordningsobjekt. Verifiera att så är fallet enligt nedan:

```
scala> Team("fnatic",1499) < Team("gurka", 2)  // Vilket fel? Varför?
scala> import Team.highestRankFirst._
scala> Team("fnatic",1499) < Team("gurka", 2)  // Inget fel? Varför?
```

Uppgift 17. *Specialanpassad ordning genom att ärva från `Ordered`.* Om det finns *en* väldefinierad, specifik ordning som man vill ska gälla för sina case-klass-instanser kan man göra den ordnad genom att låta case-klassen mixa in traiten `Ordered` och implementera den abstrakta metoden `compare`.

> *Bakgrund för kännedom:* En trait som används på detta sätt kallas **gränssnitt** (eng. *interface*), och om man *implementerar* ett gränssnitt så uppfyller man ett ”kontrakt”, som i detta fall innebär att man implementerar det som krävs av ordnade objekt, nämligen att de har en konkret `compare`-metod. Du lär dig mer om gränssnitt i kommande kurser.

a) Implementera case-klassen `Team` så att den är en subtyp till `Ordered` enligt nedan skiss. Högre rankade lag ska komma före lägre rankade lag. Metoden `compare` ska ge ett heltal som är negativt om **`this`** kommer före `that`, noll om de ordnas lika, annars positivt.

```
case class Team(name: String, rank: Int) extends Ordered[Team]{
  override def compare(that: Team): Int = ???
}
```

Tips: Du kan anropa metoden `compare` på alla grundtyper, t.ex. `Int`, eftersom de är implicit ordnade. Genom att negera uttrycket blir ordningen den omvända.

```
scala> -(2.compare(1))
```

b) Testa att din case-klass nu uppfyller det som krävs för att vara ordnad.

```
scala> Team("fnatic",1499) < Team("gurka", 2)
```

Uppgift 18. *Jämförelsestöd i Java.* Java har motsvarigheter till `Ordering` och `Ordered`, som heter `java.util.Comparator` och `java.lang.Comparable`. I själva verket så är Scalas `Ordering` en subtyp till Javas `Comparator`, medan Scalas `Ordered` är en subtyp till Javas `Comparable`.

- Javas `Comparator` och Scalas `Ordering` används för att skapa fristående ordningar som kan jämföra *två olika* objekt. I Scala kan dessa göras implicit tillgängliga. I Javas samlingsbibliotek skickas instanser av `Comparator` med som explicita argument.
- Javas `Comparable` och Scalas `Ordered` används som supertyp för klasser som vill kunna jämföra ”sig själv” med andra objekt och har *en* naturlig ordningsdefinition.

a) Sök upp dokumentationen för `java.util.Comparator`. Vilken abstrakt metod måste implementeras och vad gör den? 

b) I paketet `java.util.Arrays` finns en metod `sort` som tar en `Array[T]` och en `Comparable[T]`. Testa att använda dessa i REPL enligt nedan skiss. Starta om REPL så att ev. tidigare implicita ordningar för `Team` inte finns kvar.

```
scala> import java.util.Comparator
scala> val teamComparator = new Comparator[Team]{
         def compare(o1: Team, o2: Team) = ???
       }
scala> val xs =
         Array(Team("fnatic", 1499), Team("nip", 1473), Team("lumi", 1601))
scala> java.util.Arrays.sort(xs.toArray, teamComparator)
scala> xs
```

```
// kod till facit
val teamComparator = new Comparator[Team]{
  def compare(o1: Team, o2: Team) = o2.rank - o1.rank
}
```

c) I Scala finns en behändig metod `Ordering.comparatorToOrdering` som skapar en implicit tillgänglig ordning om man har en `java.util.Comparator`. Testa detta enligt nedan i REPL, med deklarationerna från föregående deluppgift.

```
scala> implicit val teamOrd = Ordering.comparatorToOrdering(teamComparator)
scala> xs.sorted
```

d) Sök upp dokumentationen för `java.lang.Comparable`. Vilken abstrakt metod måste implementeras och vad gör den? 

e) Gör så att klassen `Point` är `Comparable` och att punkter närmare origo sorteras före punkter som är längre ifrån origo enligt nedan skiss. I Scala är typer som är `Comparable` implicit även `Ordered`, varför sorteringen nedan funkar. Verfiera detta i REPL när du klurat ut hur implementera `compareTo`.

```
case class Point(x: Int, y: Int) extends Comparable[Point] {
  def distanceFromOrigin: Double = ???
  def compareTo(that: Point): Int = ???
}
```

```
scala> val xs = Seq(Point(10,10), Point(2,1), Point(5,3), Point(0,0))
```

```
scala> xs.sorted
```

```
// kod till facit
case class Point(x: Int, y: Int) extends Comparable[Point] {
  def distanceFromOrigin: Double = math.hypot(x, y)
  def compareTo(that: Point): Int =
    (distanceFromOrigin - that.distanceFromOrigin).round.toInt
}
```

Uppgift 19. *Fixa svensk sorteringsordning av ÄÅÖ.* Svenska bokstäver kommer i, för svenskar, konstig ordning om man inte vidtar speciella åtgärder. Med hjälp av klassen `java.text.Collator` kan man få en `Comparator` för strängar som följer lokala regler för en massa språk på planeten jorden.

a) Verifiera att sorteringsordningen blir rätt i REPL enligt nedan.

```
scala> val fel = Vector("ö","å","ä","z").sorted
scala> val svColl = java.text.Collator.getInstance(new java.util.Locale("sv"))
scala> val svOrd = Ordering.comparatorToOrdering(svColl)
scala> val rätt = Vector("ö","å","ä","z").sorted(svOrd)
```

b) Använd metoden ovan för att skriva ett program som skriver ut raderna i en textfil i korrekt svensk sorteringsordning. Programmet ska kunna köras med kommandot:
`scala sorted -sv textfil.txt`

c) Läs mer här:
stackoverflow.com/questions/24860138/sort-list-of-string-with-localization-in-scala

Uppgift 20. I klassen `java.util.Arrays`[38] finns en statisk metod `binarySearch` som kan användas enligt nedan.

```
scala> val xs = Array(5,1,3,42,-1)
scala> java.util.Arrays.sort(xs)
scala> xs
scala> java.util.Arrays.binarySearch(xs, 42)
scala> java.util.Arrays.binarySearch(xs, 43)
```

Skriv ett valfritt Javaprogram som testar `java.util.Arrays.binarySearch`. Använd en array av typen `int[]` med några heltal som först sorteras med `java.util.Arrays.sort`. Skriv ut det som returneras från `java.util.Arrays.binarySearc` i olika fall genom att asöka efter tal som finns först, mitt i, sist och tal som saknas. *Tips:* Man kan deklarera en array, allokera den och fylla den med värden så här i Java:

```
int[] xs = new int[]{5, 1, 3, 42, -1};
```

Uppgift 21. Fördjupa dig inom webbteknologi.

a) Lär dig om HTML här: `http://www.w3schools.com/html/`

[38] docs.oracle.com/javase/8/docs/api/java/util/Arrays.html

b) Lär dig om Javascript här: `http://www.w3schools.com/js/`

c) Lär dig om CSS här: `http://www.w3schools.com/css/`

d) Lär dig om Scala.JS här: `http://www.scala-js.org/`

11. Övning: scalajava

Mål

- ☐ Kunna förklara och beskriva viktiga skillnader mellan Scala och Java.
- ☐ Kunna översätta enkla algoritmer, klasser och singeltonobjekt från Scala till Java och vice versa.
- ☐ Känna till vad en case-klass innehåller i termer av en Javaklass.
- ☐ Kunna använda Javatyperna `List`, `ArrayList`, `Set`, `HashSet` och översätta till deras Scalamotsvarigheter med `JavaConverters`.
- ☐ Kunna förklara hur autoboxning fungerar i Java, samt beskriva fördelar och fallgropar.

Förberedelser

- ☐ Studera begreppen i kapitel **??**.

11.1 Grunduppgifter

Uppgift 1. *Översätta metoder från Java till Scala.* I denna uppgift ska du översätta en Java-klass som används som en modul[39] och bara innehåller statiska metoder och inget varaktigt tillstånd. (I nästa uppgift ska du sedan översätta klasser med attribut och varaktiga tillstånd.)

Vi börjar med att göra översättningen från Java till Scala rad för rad och du ska behålla så mycket som möjligt av syntax och semantik så att Scala-koden blir så Java-lik som möjligt. I efterföljande deluppgift ska du sedan omforma översättningen så att Scala-koden blir mer idiomatisk[40].

a) Studera klassen `Hangman` nedan. Du ska översätta den från Java till Scala enlig de riktlinjer och tips som följer efter koden. Läs igenom alla riktlinjer och tips innan du börjar.

```
1  import java.net.URL;
2  import java.util.ArrayList;
3  import java.util.Set;
4  import java.util.HashSet;
5  import java.util.Scanner;
6
7  public class Hangman {
8      private static String[] hangman = new String[]{
9          " ======  ",
10         " |/   |  ",
11         " |    O  ",
12         " |   -|- ",
13         " |   / \\ ",
```

[39] en.wikipedia.org/wiki/Modular_programming
[40] sv.wikipedia.org/wiki/Idiom_%28programmering%29

```
14          " |       ",
15          " |       ",
16          " ==========================   RIP  :("};
17
18      private static String renderHangman(int n){
19          StringBuilder result = new StringBuilder();
20          for (int i = 0; i < n; i++){
21              result.append(hangman[i]);
22              if (i < n - 1) {
23                  result.append("\n");
24              }
25          }
26          return result.toString;
27      }
28
29      private static String hideSecret(String secret,
30                                       Set<Character> found){
31          String result = "";
32          for (int i = 0; i < secret.length(); i++) {
33              if (found.contains(secret.charAt(i))) {
34                  result += secret.charAt(i);
35              } else {
36                  result += '_';
37              }
38          }
39          return result;
40      }
41
42      private static boolean foundAll(String secret,
43                                      Set<Character> found){
44          boolean foundMissing = false;
45          int i = 0;
46          while (i < secret.length() && !foundMissing) {
47              foundMissing = !found.contains(secret.charAt(i));
48              i++;
49          }
50          return !foundMissing;
51      }
52
53      private static char makeGuess(){
54          Scanner scan = new Scanner(System.in);
55          String guess = "";
56          do {
57             System.out.println("Gissa ett tecken: ");
58             guess = scan.next();
59          } while (guess.length() != 1);
```

```
60          return Character.toLowerCase(guess.charAt(0));
61      }
62
63      public static String download(String address, String coding){
64          String result = "lackalänga";
65          try {
66              URL url = new URL(address);
67              ArrayList<String> words = new ArrayList<String>();
68              Scanner scan = new Scanner(url.openStream(), coding);
69              while (scan.hasNext()) {
70                  words.add(scan.next());
71              }
72              int rnd = (int) (Math.random() * words.size());
73              result = words.get(rnd);
74          } catch (Exception e) {
75              System.out.println("Error: " + e);
76          }
77          return result;
78      }
79
80      public static void play(String secret){
81          Set<Character> found = new HashSet<Character>();
82          int bad = 0;
83          boolean won = false;
84          while (bad < hangman.length && !won){
85              System.out.println(renderHangman(bad));
86              System.out.print("\nFelgissningar: " + bad + "\t");
87              System.out.println(hideSecret(secret, found));
88              char guess = makeGuess();
89              if (secret.indexOf(guess) >= 0) {
90                  found.add(guess);
91              } else {
92                bad++;
93              }
94              won = foundAll(secret, found);
95          }
96          if (won) {
97              System.out.println("BRA! :)");
98          } else {
99              System.out.println("Hängd! :(");
100         }
101         System.out.println("Rätt svar: " + secret);
102         System.out.println("Antal felgissningar: " + bad);
103     }
104
105     public static void main(String[] args){
```

```
106         if (args.length == 0) {
107             String runeberg =
108                 "http://runeberg.org/words/ord.ortsnamn.posten";
109             play(download(runeberg, "ISO-8859-1"));
110         } else {
111             int rnd = (int) (Math.random() * args.length);
112             play(args[rnd]);
113         }
114     }
115 }
```

Riktlinjer och tips för översättningen:

1. Skriv Scala-koden med en texteditor i en fil som heter `hangman1.scala` och kompilera med `scalac hangman1.scala` i terminalen; använd alltså *inte* en IDE, så som Eclipse eller IntelliJ, utan en ”vanlig” texteditor, t.ex. `gedit`.
2. Översätt i denna första deluppgift rad för rad så likt den ursprungliga Java-kodens utseende (syntax) som möjligt, med så få ändringar som möjligt. Du ska alltså ha kvar dessa Scalaovanligheter, även om det inte alls blir som man brukar skriva i Scala:
   (a) långa indrag,
   (b) onödiga semikolon,
   (c) onödiga `()`,
   (d) onödiga `{}`,
   (e) onödiga `System.out`, och
   (f) onödiga **`return`**.
3. Försök också i denna deluppgift göra så att betydelsen (semantiken) så långt som möjligt motsvarar den i Java, t.ex. genom att använda **`var`** överallt, även där man i Scala normalt använder **`val`**.
4. En Javaklass med bara statiska medlemmar motsvara ett singeltonobjekt i Scala, alltså en **`object`**-deklaration innehållande ”vanliga” medlemmar.
5. För att tydliggöra att du använder Javas `Set` och `HashSet` i din Scala-kod, använd följande import-satser i `hangman1.scala`, som därmed döper om dina importerade namn och gör så att de inte krockar med Scalas inbyggda `Set`. Denna form av import går inte att göra i Java.

   ```
   import java.util.{Set => JSet};
   import java.util.{HashSet => JHashSet};
   ```

6. Javas `i++` fungerar inte i Scala; man får istället skriva `i += 1` eller mindre vanliga `i = i + 1`.
7. Typparametrar i Java skrivs inom <> medan Scalas syntax för typparametrar använder `[]`.
8. Till skillnad från Java så har Scalas metoddeklarationer ett tilldelningstecken = efter returtypen, före kroppen.
9. Du kan ladda ner Java-koden till `Hangman`-klassen nedan från kursens

repo[41]. I samma bibliotek ligger även lösningarna till översättningen i Scala, men kolla *inte* på dessa förrän du gjort klart översättningarna och fått dem att kompilera och köra felfritt! Tanken är att du ska träna på att läsa felmeddelande från kompilatorn och åtgärda dem i en upprepad kompilera-testa-rätta-cykel.

b) Skapa en ny fil `hangman2.scala` som till att börja med innehåller en kopia av din direkt-översatta Java-kod från föregående deluppgift. Omforma koden så att den blir mer som man brukar skriva i Scala, alltså mer Scala-idiomatisk. Försök förenkla och förkorta så mycket du kan utan att göra avkall på läsbarheten.

Tips och riktlinjer:

1. Kalla Scala-objektet för `hangman`. När man använder ett Scalaobjekt som en modul (alltså en samling funktioner i en gemensam, avgränsad namnrymd) har man gärna liten begynnelsebokstav, i likhet med konventionen för paketnamn. Ett paket är ju också en slags modul och med en namngivningskonvention som är gemensam kan man senare, utan att behöva ändra koden som använder modulen, ändra från ett singelobjektet till ett paket och vice versa om man så önskar.
2. Gör alla metoder publikt tillgängliga och låt även strängvektorn `hangman` vara publikt tillgänglig. Deklarera `hangman` som en **`val`** och konstruera den med `Vector`. Eftersom `Vector` är oföränderlig och man inte kan ärva från singelobjekt och `hangman` är deklarerad med **`val`** finns inga speciella risker med att göra den konstanta vektorn publik om vi inte har något emot att annan kod kan läsa (och eventuellt göra sig beroende av) vår hänggubbetext.
3. I metoden `renderHangman` använd `take` och `mkString`.
4. I metoden `hideSecret` använd `map` i stället för en **`for`**-sats.
5. Det går att ersätta metoden `findAll` med det kärnfulla uttrycket `(secret forall found)` där `secret` är en sträng och `found` är en mängd av tecken (undersök gärna i REPL hur detta fungerar). Skippa därför den metoden helt och använd det kortare uttrycket direkt.
6. I metoden `makeGuess`, i stället för `Scanner`, använd `scala.io.StdIn.readLine`.
7. Om du vill träna på att använda rekursion i stället för imperativa loopar: Gör metoden `makeGuess` rekursiv i stället för att använda **`do-while`**.
8. I metoden `download`, i stället för `java.net.URL` och `java.util.ArrayList`, använd `scala.io.Source.fromURL(address, coding).getLines.toVector` och gör en lokal import av `scala.io.Source.fromURL` överst i det block där den används. Det går inte att ha lokala **`import`**-satser i Java.
9. Låt metoden `download` returnera en `Option[String]` som i fallet att nedladdningen misslyckas returnerar `None`.
10. I metoden `download`, i stället för **`try-catch`** använd `scala.util.Try` och dess smidiga metoder `recover` och `toOption`.
11. Om du vill träna på att använda rekursion i stället för imperativa loopar: Använd, i stället för **`while`**-satsen i metoden `play`, en lokal rekursiv

[41] github.com/lunduniversity/introprog/blob/master/compendium/examples/scalajava/Hangman.java

funktion med denna signatur:

```
def loop(found: Set[Char], bad: Int): (Int, Boolean)
```

Funktionen `loop` returnerar en 2-tupel med antalet felgissningar och `true` om man hittat alla bokstäver eller `false` om man blev hängd.

Uppgift 2. *Översätta mellan klasser i Scala och klasser i Java.* Klassen `Point` nedan är en model av en punkt som kan sparas på begäran i en lista. Listan är privat för kompanjonsobjektet och kan skrivas ut med en metod `showSaved`. I koden används en `ArrayBuffer`, men i framtiden vill man, vid behov, kunna ändra från `ArrayBuffer` till en annan sekvenssamlingsimplementation, t.ex. `ListBuffer`, som uppfyller egenskaperna hos supertypen `Buffer`, men har andra prestandaegenskaper för olika operationer. Därför är attributet `saved` i kompanjonsobjektet deklarerat med den mer generella typen.

```
1  class Point(val x: Int, val y: Int, save: Boolean = false) {
2    import Point._
3
4    if (save) saved.prepend(this)
5
6    def this() = this(0, 0)
7
8    def distanceTo(that: Point) = distanceBetween(this, that)
9
10   override def toString = s"Point($x, $y)"
11 }
12
13 object Point {
14   import scala.collection.mutable.{ArrayBuffer, Buffer}
15
16   private val saved: Buffer[Point] = ArrayBuffer.empty
17
18   def distanceBetween(p1: Point, p2: Point) =
19     math.hypot(p1.x - p2.x, p1.y - p2.y)
20
21   def showSaved: Unit =
22     println(saved.mkString("Saved: ", ", ", "\n"))
23 }
```

a) Översätt klassen `Point` ovan från Scala till Java. Vi ska i nästa deluppgift kompilera både Scala-programmet ovan och ditt motsvarande Java-program i terminalen och testa i REPL att klasserna har motsvarande funktionalitet.

Tips och riktlinjer:

1. För att namnen inte ska krocka i våra kommande tester, kalla Javatypen för `JPoint`.
2. I stället för Scalas `ArrayBuffer` och `Buffer`, använd Javas `ArrayList` och `List` som båda ligger i paketet `java.util`.

3. Undersök dokumentationen för `java.util.List` för att hitta en motsvarighet till `prepend` för att lägga till i början av listan.
4. I stället för default-argumentet i Scalas primärkonstruktor, använd en extra Java-konstruktor.
5. Det finns inga singelobjekt och inga kompanjonsobjekt i Java; istället kan man använda statiska klassmedlemmar. Placera kompanjonsobjektets medlemmars motsvarigheter *innuti* Java-klassen och gör dem till **`static`**-medlemmar.
6. Kod i klasskroppen i Scalaklassen, så som if-satsen på rad 4, placeras i lämplig konstruktor i Javaklassen.
7. Utskrifter med `print` och `println` behöver i Java föregås av `System.out.`
8. Det finns inget nyckelord **`override`** i Java, men en s.k. annotering som ger samma kompilatorhjälp. Den skrivs med ett snabel-a och stor begynnelsebokstav, så här: `@Override` före metoddeklarationen.
9. I Java används konventionen att börja getter-metoder med ordet `get`, t.ex. `getX()`.
10. Det finns ingen motsvarighet till `mkString` för `List` så du behöver själv gå igenom listan och hämta elementreferenser för utskrift med en **`for`**-loop. Notera att efter sista elementet ska radbrytning göras i utskriften och att inget komma ska skrivas ut efter sista elementet.
11. I Java behövs en ny **`import`**-deklaration om man vill importera ännu en typ från samma paket. Man kan även i Java använda asterisk `*`, (motsvarande _ i Scala), för att importera allt i ett paket, men då får man med alla möjliga namn och det vill man kanske inte.
12. Metoder i Java slutar med `()` om de saknar parametrar.
13. Alla satser i Java slutar med lättglömda semikolon. (Efter att man i skrivit mycket Javakod och växlar till Scalakod är det svårt att vänja sig av med att skriva semikolon...)

b) Starta REPL i samma bibliotek som du kompilerat kodfilerna. Testa så att klasserna `Point` och `JPoint` beter sig på samma vis enligt nedan. Skriv även testkod i REPL för att avläsa de attributvärden som har getters och undersök att allt funkar som det ska.

```
$ scalac Point.scala
$ javac JPoint.java
$ scala
scala> val (p, jp) = (new Point, new JPoint)
scala> p.distanceTo(new Point(3, 4))
scala> Point.showSaved
scala> jp.distanceTo(new JPoint(3, 4))
scala> JPoint.showSaved
scala> for (i <- 1 to 10) { new Point(i, i, true) }
scala> Point.showSaved
scala> for (i <- 1 to 10) { new JPoint(i, i, true) }
scala> JPoint.showSaved
```

c) Översätt nedan Javaklass `JPerson` till en **`case class`** `Person` i Scala med motsvarande funktionalitet.

```java
public class JPerson {
    private final String name;
    private final int age;

    public JPerson(final String name, final int age){
        this.name = name;
        this.age = age;
    }

    public JPerson(final String name){
        this(name, 0);
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public boolean canEqual(Object other) {
        return (other instanceof JPerson);
    }

    @Override public boolean equals(Object other){
        boolean result = false;
        if (other instanceof JPerson) {
            JPerson that = (JPerson) other;
            result = that.canEqual(this) &&
              this.getName() == that.getName() &&
              this.getAge()  == that.getAge();
        }
        return result;
    }

    @Override public int hashCode() {
      return name.hashCode() * 41 + age;
    }

    @Override public String toString() {
      return "JPerson(" + name + ", " + age + ")";
    }
}
```

d) Undersök i REPL vilken funktionalitet i Scala-case-klassen `Person` som *in-*

te är implementerad i Java-klassen `JPerson` ovan. Skriv upp namnen på några av case-klassens extra metoder samt deras signatur genom att för en `Person`-instans, och för kompansjonsobjektet `Person`, trycka på TAB-tangenten. Prova några av de extra metoderna i REPL och förklara vad de gör.

```
scala> val p = Person("Björn", 49)
scala> p.      // tryck TAB en gång
scala> Person. // tryck TAB en gång
scala> p.copy  // tryck TAB en gång
scala> p.copy()
scala> p.copy(age = p.age + 1)
scala> Person.unapply(p)
```

Uppgift 3. *Auto(un)boxing.* I JVM måste typparametern för generiska klasser vara av referenstyp. I Scala löser kompilatorn detta åt oss så att vi ändå kan ha t.ex. `Int` som argument till en typparameter i Scala, medan man i Java *inte* direkt kan ha den primitiva typen **`int`** som typparameter till t.ex. `ArrayList`.

I Java och i den underliggande plattformen JVM används s.k. wrapper-klasser för att lösa detta, t.ex. genom wrapper-klassen `Integer` som boxar den primitiva typen **`int`**. Java-kompilatorn har stöd för att automatiskt packa in värden av primitiv typ i sådana wrapper-klasser för att skapa referenstyper och kan även automatiskt packa upp dem.

a) Studera hur Scala-kompilatorn låter oss arbeta med en `Cell[Int]` även om det underliggande JVM:ens körtidstyp (eng. *runtime type*) är en wrapper-klass. Man kan se JVM-körtidstypen med metoderna `getClass` och `getTypeName` enligt nedan.

```
scala> class Cell[T](var value: T){
         val typeName: String = value.getClass.getTypeName
         override def toString = "Cell[" + typeName + "](" + value + ")"
       }
scala> val c = new Cell[Int](42)
scala> c.value.getClass.getTypeName
```

b) Vad är körtidstypen för `c.value` ovan? Förklara hur det kan komma sig trots att vi deklarerade med typargumentet `Int`?

c) Studera dokumentationen för `java.lang.Integer`[42] och testa i REPL några av *klassmetoderna* (de som är **`static`** och därmed kan anropas med punktnotation direkt på klassens namn utan **`new`**) och några av *instansmetoderna* (de som inte är **`static`**).

```
scala> Integer.  //tryck TAB
scala> Integer.
scala> Integer.toBinaryString(42)
scala> Integer.valueOf(42)
scala> val i = new Integer(42)
scala> i.  // tryck TAB
scala> i.toString
scala> i.compareTo  // tryck TAB 2 gånger
```

[42] docs.oracle.com/javase/8/docs/api/java/lang/Integer.html

```
scala> i.compareTo(Integer.valueOf(42))
scala> i.compareTo(42)  // varför fungerar detta?
```

d) Enligt dokumentationen[43] tar instansmetoden `compareTo` i klassen `Integer` en `Integer` som parameter. Hur kan det då komma sig att sista raden ovan fungerar med en `Int`?

e) Studera nedan Java-program och beskriv vad som kommer att skrivas ut *innan* du kompilerar och testkör.

```
import java.util.ArrayList;

public class Autoboxing {
    public static void main(String[] args) {
        ArrayList<Integer> xs = new ArrayList<Integer>();
        for (int i = 0; i < 42; i++) {
            xs.add(new Integer(i));
        }
        for (Integer x: xs) {
            int y = x.intValue() * 10;
            System.out.print(y + " ");
        }
        int pos = xs.size();
        xs.add(pos, new Integer(0));
        System.out.println("\n\n[0]: " + xs.get(0).intValue());
        System.out.println("[" + pos + "]: " + xs.get(pos));
        if (xs.get(0) == xs.get(pos)) {
            System.out.println("EQUAL");
        } else {
            System.out.println("NOT EQUAL");
        }
    }
}
```

f) Ändra i programmet ovan så att autoboxing och autounboxing utnyttjas på alla ställen där så är möjligt. Utnyttja även att `toString`-metoden på `Integer` ger samma stränrepresentation som `int` vid utskrift. Fixa också så att du undviker *fallgropen* att i Java jämföra med referenslikhet i stället för att använda `equals`. Testa så att allt fungerar som det borde efter dina ändringar.

g) Antag att du råkar skriva `xs.add(0, pos)` på rad 14 i ditt program från föregående uppgift. Förklara hur autoboxingen stjälper dig i en *fallgrop* då.

h) Med ledning av de båda tidigare deluppgifterna: sammanfatta de två nämnda fallgropar med autoboxing i Java i två generella punkter, så att du har nytta av att memorera dem inför din framtida Javakodning.

[43] docs.oracle.com/javase/8/docs/api/java/lang/Integer.html#compareTo-java.lang.Integer-

Uppgift 4. *JavaConverters.* Med `import scala.collection.JavaConverters._` får man i sina Scalaprogram tillgång till de smidiga metoderna `asJava` och `asScala` som översätter mellan motsvarande samlingar i resp språks standardbibliotek. Kör nedan i REPL och gör efterföljande deluppgifter.

```
scala> val sv = Vector(1,2,3)
scala> val ss = Set('a','b','c')
scala> val sm = Map("gurka" -> 42, "tomat" -> 0)
scala> val ja = new java.util.ArrayList[Int]
scala> ja.add(42)
scala> val js = new java.util.HashSet[Char]
scala> js.add('a')
scala> import scala.collection.JavaConverters._
```

a) Till vilka typer konverteras Scalasamlingarna `Vector[Int]`, `Set[Char]` och `Map[String, Int]` om du anropar metoden `asJava` på dessa?

b) Till vilka typer konverteras Javasamlingarna `ArrayList[Int]` och `HashSet[Char]` om du anropar metoden `asScala` på dessa? Blir det föränderliga eller oföränderliga motsvarigheter?

c) Vad får resultatet för typ om du kör `toSet` på en samling av typen `mutable.Set`?

d) Undersök hur du kan efter att du gjort `sm.asJava.asScala` anropa ytterligare en metod för att få tillbaka en oföränderlig `immutable.Map`.

e) Läs mer i dokumentationen om JavaConverters[44] och prova några fler konverteringar.

11.2 Extrauppgifter

Uppgift 5. Översätt nedan kod från Java till Scala. Skriv koden i en fil som heter `showInt.scala` och kalla Scala-objektet med `main`-metoden för `showInt`. Läs tipsen som följer efter koden innan du börjar.

```
import java.util.Scanner;

public class JShowInt {
    private static Scanner scan = new Scanner(System.in);

    public static void show(Object obj) {
        System.out.println(obj);
    }

    public static void show(Object obj, String msg) {
        System.out.println(msg + obj);
    }
```

[44] docs.scala-lang.org/overviews/collections/conversions-between-java-and-scala-collections.html

```
13
14      public static String repeatChar(char ch, int n) {
15          StringBuilder sb = new StringBuilder();
16          for (int i = 0; i < n; i++) {
17              sb.append(ch);
18          }
19          return sb.toString();
20      }
21
22      public static String readLine(String prompt) {
23          System.out.print(prompt);
24          return scan.nextLine();
25      }
26
27      public static void showInt(int i) {
28          int leading = Integer.numberOfLeadingZeros(i);
29          String binaryString =
30              repeatChar('0', leading) + Integer.toBinaryString(i);
31          show(i,                         "Heltal: ");
32          show((char) i,                  "Tecken: ");
33          show(binaryString,              "Binärt: ");
34          show(Integer.toHexString(i),    "Hex   : ");
35          show(Integer.toOctalString(i), "Oktalt: ");
36      }
37
38      public static void loop() {
39          boolean hasExploded = false;
40          while (!hasExploded) {
41              try {
42                  String s = readLine("Heltal annars pang: ");
43                  showInt(Integer.parseInt(s));
44              } catch (Throwable e){
45                  show(e);
46                  hasExploded = true;
47              }
48          }
49          show("PANG!");
50      }
51
52      public static void main(String[] args){
53          if (args.length == 0) {
54              loop();
55          } else {
56              for (String arg: args) {
57                  showInt(Integer.parseInt(arg));
58                  System.out.println();
```

```
59                 }
60             }
61         }
62 }
```

Tips:

- En Javaklass med bara statiska medlemmar motsvaras av ett singeltonobjekt i Scala, alltså en **object**-deklaration. Scala har därför inte nyckelordet **static**.
- Typen `Object` i Java motsvaras av Scalas `Any`.
- Du kan använda Scalas möjlighet med default-argument (som saknas i Java) för att bara definiera en enda `show`-metod med en tom sträng som default `msg`-argument.
- I Scala har objekt av typen `Char` en metod **def** `*(n: Int): String` som skapar en sträng med tecknet repeterat `n` gånger. Men du kan ju välja att ändå implementera metoden `repeatChar` med `StringBuilder` som nedan om du vill träna på att översätta en **for**-loop från Java till Scala.
- I stället för `Scanner.nextLine` kan använda `scala.io.StdIn.readLine` som tar en prompt som parameter, men du kan också använda `Scanner` i Scala om du vill träna på det.
- I Java *måste* man använda nyckelordet **return** om metoden inte är en **void**-metod, medan man i Scala faktiskt *får* använda **return** även om man brukar undvika det och i stället utnyttja att satser i Scala också är uttryck.

Kompilera din Scala-kod och kör i terminalen och testa så att allt funkar. Vill du även kompilera Java-koden så finns den i kursens repo i filen `compendium/examples/scalajava/JShowInt.java`

Uppgift 6. TODO!!! Fallgrop med Point som inte har equals och en Polygon som kolla hasVertex som inte hittas...
https://github.com/bjornregnell/lth-eda016-2015/blob/master/lectures/examples/eclipse-ws/lecture-examples/src/week10/generics/TestPitfall3.java

11.3 Fördjupningsuppgifter

Uppgift 7. TODO!!! *Gränssnitt i Scala och Java.*

Uppgift 8. Studera fallgropar för hur man skriver en `equals`-metod i Java här: http://www.artima.com/lejava/articles/equality.html
Vilken fallgrop trillar man *inte* i om man endast överskuggar `equals` i finala klasser som inte har några superklasser?

Uppgift 9. Studera det fullständiga receptet för hur man skriver en välfungerande `equals` och `hashcode` i Scala här: http://www.artima.com/pins1ed/object-equality.html

12. Övning: `threads`

Mål

- ☐ Känna till vad en tråd är och kunna förklara begreppet jämlöpande exekvering.
- ☐ Känna till vad metoderna `run` och `start` gör i klassen `Thread`.
- ☐ Kunna skapa och starta en tråd med överskuggad `run`-metod.
- ☐ Kunna skapa ett enkelt program som från två trådar tävlar om att uppdatera en variabel och förklara varför beteendet kan bli oförutsägbart.
- ☐ Kunna använda en `Future` för att köra igång flera parallella beräkningar.
- ☐ Kunna registrera en callback på en `Future` med metoden `onComplete`.

Förberedelser

- ☐ Studera begreppen i kapitel **??**.

12.1 Grunduppgifter

Uppgift 1. *Trådar.* Klassen `java.lang.Thread` används för att skapa **trådar** med jämlöpande exekvering (eng. *concurrent execution*). På så sätt kan man få olika koddelar att köra samtidigt.

Klassen `Thread` definierar en tom `run`-metod. Vill man att tråden ska göra något vettigt får man överskugga `run` med det man vill ska göras.

En tråd körs igång med metoden `start` och då anropas automatiskt `run`-metoden och tråden exekverar koden i `run` jämlöpande med övriga trådar. Om man anropar `run` direkt blir det *inte* jämlöpande exekvering.

a) Skapa en tråd som gör något som tar lite tid och kör med `run` resp. `start`.

```
1 def zzz = { print("zzzzzz"); Thread.sleep(5000); println(" VAKEN!")}
2 zzz
3 val t2 = new Thread{ override def run = zzz }
4 t2.run
5 t2.run; println("Gomorron!")
6 t2.start; println("Gomorron!")
7 t2.start
```

b) Vad händer om man anropar `start` mer än en gång på samma tråd?

c) Skapa två trådar med överskuggade `run`-metoder och kör igång dem samtidigt enligt nedan. Vilken ordning skrivs hälsningarna ut efter rad 3 resp. rad 4 nedan? Förklara vad som händer.

```
1 val g = new Thread{ override def run = for (i <- 1 to 100) print("Gurka ") }
2 val t = new Thread{ override def run = for (i <- 1 to 100) print("Tomat ") }
3 g.run; t.run
4 g.start; t.start
```

d) Använd `Thread.sleep` enligt nedan. Är beteendet helt förutsägbart (deterministiskt)? Förklara vad som händer. Du kan (om du kör Linux) avbryta REPL med Ctrl+C[45].

```
def ibland(block: => Unit) = new Thread {
  override def run = while(true) { block; Thread.sleep(600) }
}.start
ibland(print("zzz ")); ibland(print("snark ")); ibland(println("hej!"))
```

Uppgift 2. *Jämlöpande variabeluppdatering.* Skriv klasserna `Bank` och `Kund` i en editor och klistra sedan in koden i REPL.

```
class Bank {
  private var saldo = 0;
  def visaSaldo: Unit = println(s"saldo: $saldo")
  def sättIn: Unit = { saldo += 1 }
  def taUt: Unit   = { saldo -= 1 }
}

class Kund(bank: Bank) {
  def slösaSpara = {bank.taUt; Thread.sleep(1); bank.sättIn}
}
```

a) Använd funktionen `ibland` från föregående uppgift och kör nedan rader i REPL. Resultatet av jämlöpande variabeluppdatering blir här heltokigt och leder till mycket upprörda bankkunder och -ägare. Förklara vad som händer.

```
val bank = new Bank
bank.visaSaldo
bank.sättIn
bank.visaSaldo
bank.taUt
bank.visaSaldo

val bamse = new Kund(bank)
val skutt = new Kund(bank)

bamse.slösaSpara
skutt.slösaSpara
bank.visaSaldo

def ofta(block: => Unit) = new Thread {
  override def run = while(true) { block; Thread.sleep(1) }
}.start

ofta(bamse.slösaSpara); ofta(skutt.slösaSpara)

ibland(bank.visaSaldo)
```

[45] stackoverflow.com/questions/6248884/can-i-stop-the-execution-of-an-infinite-loop-in-scala-repl

Uppgift 3. *Trådsäkra `AtomicInteger`.* Det finns stöd i JVM för att åstadkomma uppdateringar som inte kan avbrytas av andra trådar under pågående minnesskrivning. En operation som inte kan avbrytas kallas **atomär** (eng. *atomic*). Studera dokumentationen för `AtomicInteger`[46] och prova nedan kod. Förklara vad som händer.

Använd funktionerna `ofta` och `ibland` från tidigare uppgifter.

```
class SäkerBank {
  import java.util.concurrent.atomic.AtomicInteger
  private var saldo = new AtomicInteger
  def visaSaldo: Unit = println(s"saldo: ${saldo.get}")
  def sättIn: Unit = { saldo.incrementAndGet }
  def taUt: Unit   = { saldo.decrementAndGet }
}

class SäkerKund(bank: SäkerBank) {
  def slösaSpara = {bank.taUt; Thread.sleep(1); bank.sättIn}
}
```

```
1 val säkerBank = new SäkerBank
2 val farmor = new SäkerKund(säkerBank)
3 val vargen = new SäkerKund(säkerBank)
4
5 ofta(farmor.slösaSpara); ofta(vargen.slösaSpara)
6
7 ibland(säkerBank.visaSaldo)
```

Uppgift 4. *Jämlöpande exekvering med `scala.concurrent.Future`.* Att skapa och hålla reda på trådar kan bli ganska omständligt och knepigt att få rätt på. Med hjälp av `scala.concurrent.Future` kan man på ett enklare sätta skapa jämlöpande exekvering.

> *Bakgrund för kännedom:* Med en `Future` skapas jämlöpande exekvering som ”under huven” använder ett ramverk som heter Akka[47], skrivet i Scala och Java. Akka erbjuder automatisk multitrådning med s.k. trådpooler och möjliggör avancerad parallellprogrammering på en hög abstraktionsnivån, där man själv slipper skapa instanser av klassen `Thread`. I stället kan man helt enkelt placera sin kod inramat med `Future{ "körs parallellt" }` efter att man importerat det som behövs.

a) För att skapa jämlöpande exekvering med `Future` behöver man först göra import enligt nedan; då skapas ett exekveringssammanhang med trådpooler redo för användning. Starta om REPL och studera felmeddelandet efter rad 1 nedan. Importera därefter enligt nedan. Vad har `f` för typ?

```
1 scala> concurrent.Future { Thread.sleep(1000); println("En sekund senare!") }
2 scala> import scala.concurrent._
3 scala> import ExecutionContext.Implicits.global
4 scala> val f = Future { Thread.sleep(1000); println("En sekund senare!") }
```

[46] docs.oracle.com/javase/8/docs/api/java/util/concurrent/atomic/AtomicInteger.html
[47] http://akka.io/

b) Skapa en procedur `printLater` enligt nedan som skriver ut argumentet efter slumpmässigt tid. Förklara vad som händer nedan.

```
scala> def printLater(a: Any): Unit =
         Future { Thread.sleep((math.random * 10000).toInt); print(a + " ") }
scala> (1 to 42).foreach(i => printLater(i)); println("alla är igång!")
```

c) Skapa enligt nedan en `Future` som räknar ut hur många siffror det är i ett väldigt stort tal. Med `onComplete` kan man ange vad som ska göras när den tunga beräkningen är färdig; detta kallas att ”registrera en callback”. Vilken returtyp har `big`? Hur många siffror har det stora talet? Vad har `r` för typ? Justera argumentet till `big` om du inte orkar vänta på resultatet...

```
scala> BigInt(10).pow(100)
scala> BigInt(10).pow(100).toString.size
scala> def big(n: Int) = Future { BigInt(n).pow(n).toString.size }
scala> big(1234567).onComplete{r => println(r + " siffror") }
```

d) Den stora vinsten med `Future` är att man kan köra vidare under tiden, varför anropet av `Future` kallas **icke-blockerande** (eng. *non-blocking*). Det händer ibland att man ändå vill blockera exekveringen i väntan på ett resultat. Man kan då använda objektet `scala.concurrent.Await` och dess metod `result` enligt nedan. Använd `big` från föregående uppgift och gör en blockerande väntan på resultatet enligt nedan. Vad händer? Vad händer om du väntar för kort tid?

```
scala> import scala.concurrent.duration._
scala> Await.result(big(1234567), 20.seconds)
```

Uppgift 5. *Använda `Future` för att göra flera saker samtidigt.* I denna uppgift ska du ladda ner webbsidor parallellt med hjälp av `Future`, så att en nedladdning kan avslutas under tiden en annan dröjer.

a) Koden för en minimal webbsida ser ut som nedan. Du kan beskåda sidan här: `http://fileadmin.cs.lth.se/pgk/mini.html` eller skriva in nedan kod i en fil som heter något som slutar på `.html` och öppna filen i din webbläsare.

```
<!DOCTYPE html>
<html>
<body>
HELLO WORLD!
</body>
</html>
```

b) För att simulera slöa webbservrar kan man ladda ner en sida via sajten `http://deelay.me/`. Ladda ner ovan sida med 2 sekunders fördröjning:
`http://deelay.me/2000/http://fileadmin.cs.lth.se/pgk/mini.html`

c) Man kan ladda ner webbsidor med `scala.io.Source`. Vad händer nedan? Försök, med ledning av hur `delay` beräknas, uppskatta hur lång tid du måste

vänta i medeltal, i bästa fall, respektive värsta fall, innan du kan se första webbsidan i vektorn `laddningar` nedan?

```
scala> def ladda(url: String) = scala.io.Source.fromURL(url).getLines.toVector
scala> def slöladda(url: String) = {
         val delay = (math.random * 1000 + 2000).toInt
         val delaySite = s"http://deelay.me/$delay/"
         ladda(delaySite+url)
       }
scala> ladda("http://fileadmin.cs.lth.se/pgk/mini.html")
scala> def seg = slöladda("http://fileadmin.cs.lth.se/pgk/mini.html")
scala> val laddningar = Vector.fill(10)(seg)
scala> laddningar(0)
```

d) Innan vi kan köra igång en `Future` så måste vi, som visats i uppgift 4 importera den underliggande exekveringsmiljön som är redo att parallelisera ditt program i trådar utan att du själv måste skapa dem. Vad händer nedan?

```
scala> import scala.concurrent._
scala> import ExecutionContext.Implicits.global
scala> val f = Future{ seg }
scala> f   // kolla om den är klar annars prova igen senare
scala> f
```

e) Ladda indata utan att blockera (eng. *non-blocking input*). Förklara vad som händer nedan.

```
scala> val nonblock = Future{ Vector.fill(10)(seg) }
scala> nonblock   // kolla igen senare om ej klar
scala> nonblock
```

f) Ladda indata separat i olika parallella trådar. Förklara vad som händer nedan. Kör uttrycket på rad 3 nedan upprepade gånger i snabb följd efter varandra med pil-upp+Enter i REPL.

```
scala> val para = Vector.fill(10)(Future{ seg })
scala> para
scala> para.map(_.isCompleted)
scala> para.map(_.isCompleted) // studera hur de blir färdiga en efter en
scala> para(0)
```

g) Registrera en callback med metoden `onComplete`. Förklara vad som händer nedan.

```
scala> val action = Vector.fill(10)(Future{ seg })
scala> action(0).onComplete(xs => println(s"ready:$xs"))
scala> // vänta tills laddning på plats 0 är klar
```

h) Registrera en callback för felhantering i händelse av undantag med metoden `onFailure`. Förklara vad som händer nedan.

```
scala> def lycka  = { Thread.sleep(3000); println(":)") }
scala> def olycka = { Thread.sleep(3000); 42 / 0; lycka }
scala> Future{ lycka  }.onFailure{ case e => println(s":( $e") }
scala> Future{ olycka }.onFailure{ case e => println(s":( $e") }
```

12.2 Extrauppgifter

Uppgift 6. Räkna ut stora primtal parallellt genom att använda nedan funktioner. Implementera `isPrime` enligt pseudokod från den engelska wikipediasidan om primtalstest[48] med den s.k. ”naiva algoritmen”. Räkna ut 10 st slumpvisa primtal med 16 siffror vardera. Gör beräkningarna parallellt med hjälp av `Future`.

```
def isPrime(n: BigInt): Boolean = ???

def nextPrime(start: BigInt): BigInt = {
  var i = start
  while (!isPrime(i)) { i += 1 }
  i
}

def randomBigInt(nDigits: Int): BigInt = {
   def rndChar = ('0' + (math.random * 10).toInt).toChar
   val str = Array.fill(nDigits)(rndChar).mkString
   BigInt(str)
}
```

✎ **Uppgift 7.** *Svara på teorifrågor.*

a) Vad är en tråd?

b) Hur skapar man en tråd med klassen `Thread`?

c) Hur startar man en tråd?

d) Vilka problem kan man råka ut för om man uppdaterar samma resurs i flera olika trådar?

e) Vad innbär det att kod är *trådsäker*?

f) Nämn några fördelar med att använda Future jämfört med att använda trådar direkt.

Uppgift 8. Läs om och testa klasserna AtomicBoolean, AtomicDouble och AtomicReferens för atomär uppdatering i paketet `java.util.concurrent.atomic`. Använd några av dessa tillsammans med `scala.concurrent.Future`.

[48] en.wikipedia.org/wiki/Primality_test

12.3 Fördjupningsuppgifter

Uppgift 9. *Skapa din egen multitrådade webbserver.*

a) Skriv in[49] nedan kod i en editor och spara i en fil med namn `webserver.scala` och kompilera och kör med `scala webserver.start` och beskriv vad som händer när du med din webbläsare surfar till adressen:
`http://localhost:8089/abbasillen`

```
1  package webserver
2
3  import java.net.{ServerSocket, Socket}
4  import java.io.OutputStream
5  import java.util.Scanner
6  import scala.util.Try
7
8  object html {
9    def page(body: String): String = //minimal web page
10     s"""<!DOCTYPE html>
11        |<html>
12        |<head><meta charset="UTF-8"><title>Min Sörver</title></head>
13        |<body>
14        |$body
15        |</body>
16        |</html>
17        """.stripMargin
18
19   def header(length: Int): String = //standardized header of reply
20     s"HTTP/1.0 200 OK\nContent-length: $length\n" +
21      "Content-type: text/html\n\n"
22 }
23
24
25 object start {
26   def handleRequest(cmd: String, uri: String, socket: Socket): Unit = {
27     val os = socket.getOutputStream
28     val response = html.page("Baklänges: " + uri.reverse)
29     os.write(html.header(response.size).getBytes("UTF-8"))
30     os.write(response.getBytes("UTF-8"))
31     os.close
32     socket.close
33   }
34
35   def serverLoop(server: ServerSocket): Unit = {
36     println(s"http://localhost:${server.getLocalPort}/hej")
37     while (true) {
38       Try {
39         var socket = server.accept  // blocks thread until connect
40         val scan = new Scanner(socket.getInputStream, "UTF-8")
41         val (cmd, uri) = (scan.next, scan.next)
42         println(s"Request: $cmd $uri")
43         handleRequest(cmd, uri, socket)
```

[49] Eller ladda ner här: github.com/lunduniversity/introprog/blob/master/compendium/examples/simple-web-server/webserver.scala

```
44        }.recover{ case e: Throwable => s"Connection failed: $e" }
45      }
46    }
47
48    def main(args: Array[String]) {
49      val port = Try{ args(0).toInt }.getOrElse(8089)
50      serverLoop(new ServerSocket(port))
51    }
52  }
```

b) Du ska nu skapa en webbserver som gör något lite mer intressant. Den ska svara om du surfar till `http://localhost:8089/fib/13` med det 13:e Fibbonnaci-talet[50]. Spara din webserver från föregående deluppgift under det nya namnet `fibserver.scala` och använd koden nedan och lägg till och ändra så att din server kan svara med Fibonaccital. Vi börjar med att räkna ut Fibonaccital i funktionen `compute.fib` nedan på ett onödigt processorkrävande sätt med exponentiell tidskomplexitet så att webbservern verkligen får jobba, för att i senare deluppgifter implementera `compute.fib` med linjär tidskomplexitet och därmed undvika onödig planetuppvärmning.

```
object compute {
  def fib(n: BigInt): BigInt = {
    if (n < 0) 0 else
    if (n == 1 || n == 2) 1
    else fib(n - 1) + fib(n -2)
  }
}

def fibResponse(num: String) = Try { num.toInt } match {
  case Success(n) => html.page(s"fib($n) == " + compute.fib(n))
  case Failure(e) => html.page(s"FEL $e: skriv heltal, inte $num")
}

def errorResponse(uri:String) = html.page("FATTAR NOLL: " + uri)

def handleRequest(cmd: String, uri: String, socket: Socket): Unit = {
  val os = socket.getOutputStream
  val parts = uri.split('/').drop(1) // skip initial slash
  val response: String = (parts.head, parts.tail) match {
    case (head, Array(num)) => fibResponse(num)
    case _                  => errorResponse(uri)
  }
  os.write(html.header(response.size).getBytes("UTF-8"))
  os.write(response.getBytes("UTF-8"))
  os.close
  socket.close
}
```

Kör i terminalen med `scala fibserver.start` och beskriv vad som händer i din webbläsare när du surfar till servern.

[50] https://sv.wikipedia.org/wiki/Fibonaccital

c) Surfa efter flera stora Fibonacci-tal samtidigt i olika flikar i din browser. Hur märks det att servern bara kör i en enda tråd?

d) Gör din server multitrådad med hjälp av den nya server-loopen nedan.

```
import scala.concurrent._
import ExecutionContext.Implicits.global

  def serverLoop(server: ServerSocket): Unit = {
    println(s"http://localhost:${server.getLocalPort}/hej")
    while (true) {
      Try {
        var socket = server.accept  // blocks thread until connect
        val scan = new Scanner(socket.getInputStream, "UTF-8")
        val (cmd, uri) = (scan.next, scan.next)
        println(s"Request: $cmd $uri")
        Future { handleRequest(cmd, uri, socket) }.onFailure {
          case e => println(s"Reqest failed: $e")
        }
      }.recover{ case e: Throwable => s"Connection failed: $e" }
    }
  }
```

e) Surfa efter flera stora Fibonacci-tal samtidigt i olika flikar i din browser. Hur märks det att servern är multitrådad?

f) Det är onödigt att räkna ut samma Fibonacci-tal flera gånger. Med hjälp av en cache i form av en föränderlig `Map` kan du spara undan redan uträknade värden. Det funkar dock inte med en vanlig `scala.collection.mutable.Map` i vår multitrådade webbserver, eftersom den inte är **trådsäker** (eng. *thread-safe*). Med trådosäkra föränderliga datastrukturer blir det samma besvärliga beteende som i uppgift 2.

Du ska i stället använda `java.util.concurrent.ConcurrentHashMap`. Sök upp dokumentationen för `ConcurrentHashMap` och försök förstå koden nedan. Hur fungerar metoderna `containsKey`, `put` och `get`?

```
object compute {
  import java.util.concurrent.ConcurrentHashMap
  val memcache = new ConcurrentHashMap[BigInt, BigInt]

  def fib(n: BigInt): BigInt =
    if (memcache.containsKey(n)) {
      println("CACHE HIT!!! no need to compute: " + n)
      memcache.get(n)
    } else {
      println("cache miss :( must compute fib:  " + n)
      val f = fastFib(n)
      memcache.put(n, f)
      f
    }

  private def fastFib(n: BigInt): BigInt = {
```

```
    if (n < 0) 0 else
    if (n == 1 || n == 2) 1
    else fib(n - 1) + fib(n -2)
  }
}
```

g) Använd ovan `fib`-objekt i en ny version av din webserver. Spara den i en ny kodfil med namnet `fibserver-memcached.scala`. Undersök hur snabbt det går med stora Fibonaccital med den nya varianten. Hur stora tal kan du räkna ut? Kan servern fortsätta efter överflödad stack? Förklara varför.

h) Nu när vi kan få väldigt stora Fibonacci-tal kan det vara användbart att stoppa in radbrytningar på webbsidan. Html-taggen `</br>` ger en radbrytning.

```
  def insertBreak(s: String, n: Int = 80): String = {
    if (s.size < n) s
    else s.take(n) + "</br>" + insertBreak(s.drop(n),n)
  }
```

Använd den rekursiva funktionen ovan för att pilla in radbrytningstaggar på var n:te position i långa strängar. Testa hur det ser ut på webbsidan med ovan funktion när din server svarar med väldigt stora tal.

i) Vi ska nu använda det större heap-minnet i stället för stack-minnet och därmed inte begränsas av stackens max-storlek. Skriv om `fastFib` så att den använder en **`while`**-sats i stället för ett rekursivt anrop. Denna uppgift är ganska klurig, men om du kör fast kan du snegla i lösningarna i Appendix för inspiration.

Hur stora tal klarar din server nu? Vad händer med servern när minnet tar slut? Hur kan du skydda servern så att den inte kan hänga sig?

Uppgift 10. Utöka din server med fler beräkningsintensiva funktioner. Exempelvis primtalsberäkningar eller beräkningar av valfritt antal decimaler av π eller e. Utnyttja gärna det du lärt dig i matematiken om summor och serieutvecklingar.

Uppgift 11. Läs mer om `Future` och jämlöpande exekvering i Scala här:
alvinalexander.com/scala/future-example-scala-cookbook-oncomplete-callback

Uppgift 12. Läs mer om jämlöpande exekvering och multitrådade program i Java här: www.tutorialspoint.com/java/java_multithreading.htm
När man skriver program med jämlöpande exekvering finns det många fallgropar; det kan bli kapplöpning (eng. *race conditions*) om gemensamma resurser och dödläge (eng. *deadlock*) där inget händer för att trådar väntar på varandra. Mer om detta i senare kurser.

✎ **Uppgift 13.** *Studera dokumentationen i* `scala.concurrent`.

a) Studera dokumentationen för `scala.concurrent.Future`[51]. Hur samverkar `Future` med `Try` och `Option`? Vilka vanliga samlingsmetoder känner du igen?

b) Studera dokumentationen för `scala.concurrent.duration.Duration`[52]. Vilka tidsenheter kan användas?

c) Vid import av `scala.concurrent.duration._` dekoreras de numeriska klasserna med metoder för att skapa instanser av klassen `Duration`. Detta möjliggörs med hjälp av klassen `scala.concurrent.duration.DurationConversions`. Studera dess dokumentation och testa att i REPL skapa några tidsperioder med metoderna på `Int`.

Uppgift 14. Fördjupa dig inom webbteknologi.

a) Lär dig om HTML här: `http://www.w3schools.com/html/`

b) Lär dig om Javascript här: `http://www.w3schools.com/js/`

c) Lär dig om CSS här: `http://www.w3schools.com/css/`

d) Lär dig om Scala.JS här: `http://www.scala-js.org/`

[51] http://www.scala-lang.org/files/archive/api/current/#scala.concurrent.Future
[52] www.scala-lang.org/api/current/#scala.concurrent.duration.Duration