Programmering, grundkurs

Övningar

Utkast – pågående arbete

Kommentarer välkomna:
bjorn.regnell@cs.lth.se

Datavetenskap, LTH
Lunds Universitet

https://github.com/lunduniversity/introprog

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`

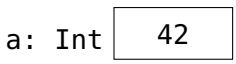
e) `true || true`

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`.)

1.3 Fördjupningsuppgifter

Uppgift 26. 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 27. TODO!!! Backticks. `val konstig val = 42`

Uppgift 28. TODO!!! Integer.toBinaryString, Integer.toHexString

Uppgift 29. TODO!!! Typannoteringar.

Uppgift 30. TODO!!! 0x2a

Uppgift 31. TODO!!! `i += 1; i *= 1; i /= 2`

Uppgift 32. TODO!!! BigInt, BigDecimal

Uppgift 33. TODO!!! Vad händer här?

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

Uppgift 34. Sök reda på dokumentationen i javadoc för klassen `java.lang.Math` i JDK 8. Tryck Ctrl+F i webbläsaren och sök efter förekomster av texten *"overflow"*. Vad är "overflow"? Vilka metoder finns i `java.lang.Math` som hjälper dig att upptäcka om det blir overflow?

Uppgift 35. 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 36. 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 37. 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 med hjälp av en editor en fil med namn `hello-script.scala` som innehåller denna enda rad:

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

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

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

a) Vad händer?

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

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

```
> 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 initialiseras 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. TODO!!! ArrayBuffer vs Vector vs Array och metoden append

Uppgift 19. Läs om krullparenteser och vanliga parenteser på stack overflow: what-is-the-formal-difference-in-scala-between-braces-and-parentheses-and-when

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. TODO!!! Visa anropsstacken genom att kasta undantag.

3.3 Fördjupningsuppgifter

Uppgift 21. *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 22. *Undersök svansrekursion genom att kasta undantag.* Förklara vad som händer. Kan du hitta bevis för att kompilatorn kan optimera rekursionen till en vanlig loop?

```
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 23. *`@tailrec`-annotering.* Du kan be kompilatorn att ge felmeddelande om den inte kan optimera koden till en loop och därmed öka prestanda och unvika 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:

```
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 skaps med `new` kan placera attributen i en parameterlista till klassen. Koden som körs när objektet skapas och attributen tilldelas sina initaialvärden, 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 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)
```

```
3         def add(p: Point): Point = Point(x + p.x, y + p.y)
4         def +(p: Point): Point = add(p)
5       }
```

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?

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

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")
```

```
4 scala> val stor = Vector.fill(100000)(math.random)
```

Undersök i REPL vad som händer nedan. Alla dessa metoder fungerar på alla samlingar som är indexerbara sekvenser. Givet deklarationerna ovan: vad har uttrycken nedan för värde och typ? Förklara vad som händer 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))
```

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`.

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

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

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.

```
scala> telnr.toVector.map(p => ???)
res85: Vector[(String, String)] = Vector(("Björn", "0462229009"), ("Maj",
"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.

4.2 Extrauppgifter

Uppgift 23. TODO!!! Träna mer på klass

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

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

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

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

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

Uppgift 24. TODO!!! Träna mer på mängd

a) TODO!!! Keno-bollar.

4.3 Fördjupningsuppgifter

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

Uppgift 26. *Dokumentationen för samlingar.* Leta upp metoden `tabulate` i dokumentationen för objektet `Vector` nästan längst ner i listan här:
http://www.scala-lang.org/api/current/#scala.collection.immutable.Vector$
Leta upp den variant av `tabulate` som har signaturen:
`def tabulate[A](n: Int)(f: (Int) => A): Vector[A]`
Klicka på den gråfyllda trekanten till vänster om signaturen som fäller ut beskrivningen

a) Förklara vad som händer här:

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

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

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

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

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

a) `b.indexWhere(s => s.startsWith("b"))`

b) `a.indices`

c) `xs.patch(1, Vector(42,43,44), 7)`

d) `xs.segmentLength(_ < 8, 2)`

e) `b.sortBy(_.reverse)`

f) `b.sortWith((s1, s2) => s1.size < s2.size)`

g) `a.reverseMap(_.size)`

h) `a intersect Vector("ka", "boom", "pow")`

i) `a diff Vector("ka")`

j) `a union Vector("ka", "boom", "pow")`

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

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

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

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

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?

```
scala> def printAll(xs: Int*) = xs.foreach(println)
scala> printAll(42)
scala> printAll(1, 2, 7, 42)
scala> def printStrings(wa: String*) = println(wa)
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 ← antalet element i xs
2 ys ← en ny heltalsarray med plats för n element
3 i ← 0
4 while i < n do
5 |   ys(n − i − 1) ← xs(i)
6 |   i ← 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 ← en ny StringBuilder som innehåller s
2 n ← antalet tecken i s
3 for i ← 0 to n/2 − 1 do
4 |   temp ← sb(i)
5 |   sb(i) ← sb(n − i − 1)
6 |   sb(n − i − 1) ← 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.

5.3 Fördjupningsuppgifter

Uppgift 19. 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 20. 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 21. 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 22. TODO!!! Keno-dragningar under ett år -> Registrering...

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] 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.

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

```
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 = ???

  /** 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
```