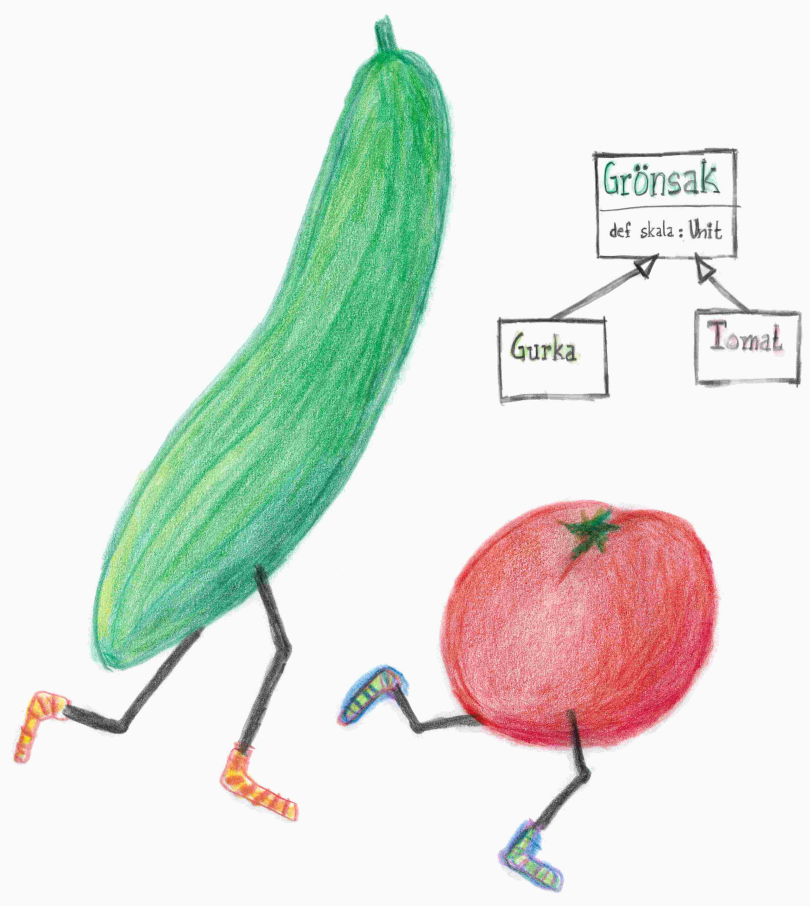
Introduktion till programmering med Scala

Lösningar till övningar



EDAA45, Lp1-2, HT 2023
Datavetenskap, LTH
Lunds Universitet

Kompileringsdatum: 7 november 2023
http://cs.lth.se/pgk

Innehåll

1. Lösning `expressions`

1.1 Grunduppgifter; förberedelse inför laboration

Lösn. uppg. 1. *Para ihop begrepp med beskrivning.*

litteral	1	⇝	D	anger ett specifikt datavärde
sträng	2	⇝	G	en sekvens av tecken
sats	3	⇝	F	en kodrad som gör något; kan särskiljas med semikolon
uttryck	4	⇝	H	kombinerar värden och funktioner till ett nytt värde
funktion	5	⇝	K	vid anrop beräknas ett returvärde
procedur	6	⇝	J	vid anrop sker (sido)effekt; returvärdet är tomt
exekveringsfel	7	⇝	N	kan inträffa medan programmet kör
kompileringsfel	8	⇝	M	kan inträffa innan exekveringen startat
abstrahera	9	⇝	A	att införa nya begrepp som förenklar kodningen
kompilera	10	⇝	C	att översätta kod till exekverbar form
typ	11	⇝	I	beskriver vad data kan användas till
for-sats	12	⇝	O	bra då antalet repetitioner är bestämt i förväg
while-sats	13	⇝	P	bra då antalet repetitioner ej är bestämt i förväg
tilldelning	14	⇝	L	för att ändra en variabels värde
flyttal	15	⇝	E	decimaltal med begränsad noggrannhet
boolesk	16	⇝	B	antingen sann eller falsk

Lösn. uppg. 2. *Utskrift i Scala REPL.*

a) Till exempel:

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

b) Om högerparentes fattas får man fortsätta skriva på nästa rad. Detta indikeras med vertikalstreck i början av varje ny rad:

```
scala> println("hejsan svejsan"
     | + "!"
     | )
hejsan svejsan!
```

Lösn. uppg. 3. *Konkatenering av strängar.*

a)

```
scala> "gurk" + "burk"
res1: String = gurkburk
```

värde: `"gurkburk"`, typ: `String`

b)

```
scala> res1 * 42
res2: String = gurkatomatgurkatomatgurkatomatgurkatomatgurkatomatgurkatomatgurk
```

Lösn. uppg. 4. *När upptäcks felet?*

a) Typ: `String`, värde: `"hejhejhej"`

b) Körtiddsfel:

```
scala> "hej" * Int.MaxValue
java.lang.OutOfMemoryError: Java heap space
```

c) Kompileringsfel: (indikeras av texten `<console> ... error:`)

```
scala> "hej" * true
<console>:12: error: type mismatch;
 found   : Boolean(true)
 required: Int
       "hej" * true
```

Ett typfel innebär att kompilatorn inte kan få typerna att överensstämma i t.ex. ett funktionsanrop. I Scala får vi reda på typfel redan vid kompilering medan i andra språk (t.ex. Javascript) upptäcks sådana fel under exekveringen, i värsta fall genom svårhittade buggar som kanske först märks långt senare.

Lösn. uppg. 5. *Litteraler och typer.*

a)

`1`	1	⇝	E	`Int`
`1L`	2	⇝	G	`Long`
`1.0`	3	⇝	J	`Double`
`1D`	4	⇝	F	`Double`
`1F`	5	⇝	H	`Float`
`'1'`	6	⇝	I	`Char`
`"1"`	7	⇝	A	`String`
`true`	8	⇝	C	`Boolean`
`false`	9	⇝	B	`Boolean`
`()`	10	⇝	D	`Unit`

b) Värdet går över gränsen för vad som får plats i ett 32 bitars heltal och ”börjar om” på det minsta möjliga heltalet `Int.MinValue` eftersom det är så binär aritmetik aritmetik med begränsat antal bitar fungerar i CPU:n.

```
1 scala> Int.MaxValue + 1
2 res3: Int = -2147483648
3
4 scala> Int.MinValue
5 res4: Int = -2147483648
```

c) Båda är heltal men `Long` kan representera större tal än `Int`.

d) Båda är flyttal men `Double` har dubbel precision och kan representera större tal med fler decimaler.

Lösn. uppg. 6. *Matematiska funktioner. Använda dokumentation.*

a) Beräkning av $2^{64} - 1$ med `math.pow` enligt nedan ger ungefär $1.8 \cdot 10^{19}$

```
scala> math.pow(2, 64) - 1
res0: Double = 1.8446744073709552E19
```

b) Ja, returtyp-annoteringen `: Double` kan utelämnas.

- Varför kan returtyp utelämnas?
  Eftersom kompilatorns typhärledning kan härleda returtypen.
- Varför kan man vilja utelämna den?
  Det blir kortare att skriva utan.
- Anledningar att ange returtyp:
  - Med explicit returtyp får du hjälp av kompilatorn att redan under kompileringen kontrollera att uttrycket till höger om likhetstecknet har den typ som förväntas.
  - Genom att du anger returtypen explicit får de som enbart läser metodhuvudet (och inte implementationen) tydligt se vad som returneras.

c) Ca 500 km.

```
scala> omkrets(12750 / 2) / 80
res0: Double = 500.6913291658733
```

Lösn. uppg. 7. *Variabler och tilldelning. Förändringsbar och oföränderlig variabel.*

a)

Efter rad 1: `a: Int` 13

Efter rad 2: `a: Int` 13 `b: Int` 14

Efter rad 3: `a: Int` 13 `b: Int` 14 `c: Double` 54.0

Efter rad 4: `a: Int` 13 `b: Int` 14 `c: Double` 54.0

Efter rad 5: `a: Int` 0 `b: Int` 14 `c: Double` 54.0

Efter rad 6: `a: Int` 0 `b: Int` 14 `c: Double` 55.0

b) Oföränderliga variabler deklareras med nyckelordet **`val`**. Det går inte att tilldela en oföränderlig variabel ett nytt värde; vid försök blir det kompileringsfel som lyder **`error: reassignment to val`**. Kompileringsfel känns igen med hjälp av texten **`error:`**, så som visas nedan:

```
scala> b = 0
<console>:12: error: reassignment to val
       b = 0
         ^
```

Lösn. uppg. 8. *Slumptal med* `math.random()`.

a) Ur dokumentationen:

```
/** Returns a Double value with a positive sign,
 *  greater than or equal to 0.0 and less than 1.0.
 */
def random(): Double
```

Dokumentationskommentarer, som börjar med `/**` och slutar med `*/`, ger oss en beskrivning av hur funktionen fungerar. Efter dokumentationskommentaren kommer funktionshuvudet, som här berättar att funktionen heter `random` och alltid kommer att returnera en `Double`. (Verktyget `scaladoc` kan med hjälp av dokumentationskommentarerna automatiskt generera webbsajter med speciella dokumentationssidor och sökfunktioner.)

b)

```
scala> def roll: Int = (math.random() * 6 + 1).toInt

scala> roll
res0: Int = 4

scala> roll
res1: Int = 1
```

Lösn. uppg. 9. *Repetition med* `for`, `foreach` *och* `while`.

a)

```
for i <- 1 to 100 do print(s"$roll, ")
```

b)

```
(1 to 100).foreach(i => print(s"$roll, "))
```

c)

```
var i = 1
while i <= 100 do { print(s"$roll, "); i = i + 1 }
```

```
var i = 1
while i <= 100 do
    print(s"$roll, ")
    i += 1
```

Lösn. uppg. 10. *Alternativ med* ***if****-sats och* ***if****-uttryck.*

a)

```
for i <- 1 to 100 do
  if roll == 6 then print("GRATTIS! ") else print(":(")
```

eller

```
for (i <- 1 to 100) if (roll == 6) print("GRATTIS! ") else print(":(")
```

b)

```
var i = 1
var n = 0
while i <= 100 do
  if roll == 6 then n = n + 1
  i = i + 1
println("Antalet sexor: " + n)
```

Lösn. uppg. 11. *Sekvens, sats och block.*

a) Satserna skapar denna utskrift:

```
san!hej
san!hej
san!hej
san!hej
```

b)

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

c)

- Klammerparenteser används för att gruppera flera satser. Klammerparenteser behövs om man vill definiera en funktion som består av mer än en sats. Sedan scala 3 kan man istället använda indentering för att definera en funktion med flera rader och satser.
- Semikolon särskiljer flera satser. Semikolon behövs om man vill skriva många satser på samma rad.

Lösn. uppg. 12. *Heltalsdivision.*

`4 / 42`	1	⇝	F	`0: Int`
`42.0 / 2`	2	⇝	C	`21.0: Double`
`42 / 4`	3	⇝	B	`10: Int`
`42 % 4`	4	⇝	G	`2: Int`
`4 % 42`	5	⇝	A	`4: Int`
`40 % 4 == 0`	6	⇝	D	`true : Boolean`
`42 % 4 == 0`	7	⇝	E	`false: Boolean`

Lösn. uppg. 13. *Booleska värden.*

a) `true`
b) `false`
c) `true`
d) `true`
e) `false`
f) `true`
g) `true`
h) `true`
i) Undantag kastas: `java.lang.ArithmeticException: / by zero`
j) `false`

Lösn. uppg. 14. *Booleska variabler.*

2: Ingenting skrivs ut.

4: `akta dig!!!`

Lösn. uppg. 15. *Turtle graphics med Kojo.*

a) Genom att börja din Kojo-program med `sudda` så startar du exekveringen i samma utgångsläge: en tom rityta (eng. *canvas*) där paddan pekar uppåt, pennan är nere och pennans färg är röd. Då blir det lättare att resonera om vad programmet gör från början till slut, jämfört med om exekveringen beror på resultatet av tidigare exekveringar.

b)

```
sudda

fram; vänster
fram; vänster
fram; vänster
fram; vänster
```

c)

```
sudda

fram; vänster
fram; höger

fram; vänster
fram; höger

fram; vänster
fram; höger

fram; vänster
```

1.2 Extrauppgifter; träna mer

Lösn. uppg. 16. *Typ och värde.*

`1.0 + 18`	1	⇝	H	`19.0: Double`
`(41 + 1).toDouble`	2	⇝	K	`42.0: Double`
`1.042e42 + 1`	3	⇝	A	`1.042E42: Double`
`12E6.toLong`	4	⇝	I	`12000000: Long`
`32.toChar.toString`	5	⇝	E	`" ": String`
`'A'.toInt`	6	⇝	B	`65: Int`
`0.toInt`	7	⇝	F	`0: Int`
`'0'.toInt`	8	⇝	D	`48: Int`
`'9'.toInt`	9	⇝	L	`57: Int`
`'A' + '0'`	10	⇝	C	`113: Int`
`('A' + '0').toChar`	11	⇝	J	`'q': Char`
`"*!%#".charAt(0)`	12	⇝	G	`'*': Char`

Lösn. uppg. 17. *Satser och uttryck.*

a) Ett utryck kan evalueras och resulterar då i ett användbart värde. En sats *gör* något (t.ex. skriver ut något), men resulterat inte i något användbart värde.

b) `println()`

c)
värdeSaknas innehåller Unit
Skriver ut `Unit`
Skriver ut `"()"`
Skriver ut `"()"`
Skriver först ut hej med det innersta anropet och sen `()` med det yttre anropet

d) `Unit`

e) `Unit`

Lösn. uppg. 18. *Procedur med parameter.*

a)

```
var highscore = 0
```

b)

```
def updateHighscore(points: Int): Unit =
  if points > highscore then
    highscore = points
    println("REKORD!")
  else println("GE INTE UPP!")
```

c)

```
def updateHighscore(points: Int): String =
```

```
if points > highscore then
  highscore = points
  "REKORD!"
else "GE INTE UPP!"
```

Lösn. uppg. 19. *Flyttalsaritmetik.*

a)

```
scala> Double.MinPositiveValue
res0: Double = 4.9E-324
```

b)

```
scala> Double.MaxValue + Double.MinPositiveValue == Double.MaxValue
res2: Boolean = true
```

Lösn. uppg. 20. ***if****-sats.*

1. Utskrift: `falskt`
2. Utskrift: `sant`
3. Inget skrivs ut, funktionen deklareras men körs ej.
4. Utskrift: `1:krona 2:klave 3:krona 4:krona 5:klave` eller liknande beroende på vilka slumptal `math.random()` ger.

Lösn. uppg. 21. ***if****-uttryck.* Notera typen `Any` på de sista två uttrycken.

```
scala> if grönsak == "tomat" then "gott" else "inte gott"
res0: String = inte gott

scala> if frukt == "banan" then "gott" else "inte gott"
res1: String = gott

scala> if true then grönsak else 42
res2: Any = gurka

scala> if false then grönsak else 42
res3: Any = 42
```

Lösn. uppg. 22. *Modulo-operatorn* `%` *och Booleska värden.*

a)

```
scala> def isEven(n: Int): Boolean = n % 2 == 0

scala> isEven(42)
res0: Boolean = true

scala> isEven(43)
res1: Boolean = false
```

b)

```
scala> def isOdd(n: Int): Boolean = !isEven(n)

scala> isOdd(42)
res2: Boolean = false

scala> isOdd(43)
res3: Boolean = true
```

Lösn. uppg. 23. *Skillnader mellan* ***var***, ***val***, ***def***.

a)

```
  scala> var x = 30
  x: Int = 30

  scala> x + 1
  res6: Int = 31

  scala> x = x + 1
  x: Int = 31

  scala> x == x + 1
  res7: Boolean = false

  scala> val y = 20
  y: Int = 20

  scala> y = y + 1
  <console>:12: error: reassignment to val
         y = y + 1
           ^

  scala> var z = { println("hej z!"); math.random() }
  hej z!
  z: Double = 0.3381365875903367

  scala> def w = { println("hej w!"); math.random() }
  w: Double

  scala> z
  res8: Double = 0.3381365875903367

  scala> z
  res9: Double = 0.3381365875903367

  scala> z = z + 1
  z: Double = 1.3381365875903368

  scala> w
  hej w!
  res10: Double = 0.06420209879434557

  scala> w
  hej w!
  res11: Double = 0.5777951341051852
```

```
scala> w = w + 1
<console>:12: error: value w_= is not a member of object
       w = w + 1
```

b)

- `var namn = uttryck` används för att deklarera en förändringsbar variabel. Namnet kan med hjälp av en tilldelningssats referera till nya värden.
- `val namn = uttryck` används för att deklarera en oföränderlig variabel som efter initialisering inte kan förändras med tilldelningssatser. Vid försök ges kompileringsfel.
- `def namn = uttryck` används för att deklarera en funktion vars uttryck evalueras varje gång den anropas.

Lösn. uppg. 24. *Skillnaden mellan* ***`if`*** *och* ***`while`***.

- Rad 3: Har du tur (50% chans) får du vinst en gång.
- Rad 4: Har du tur får du många vinster i rad. Sannolikheten för n vinster i rad är $(\frac{1}{2})^n$.

1.3 Fördjupningsuppgifter; utmaningar

Lösn. uppg. 25. *Logik och De Morgans Lagar.*

a) `poäng > 1000`

b) `poäng > 100`

c) `poäng <= highscore`

d) `poäng <= 0 || poäng >= highscore`

e) `poäng >= 0 && poäng <= highscore`

f) `klar`

g) `!klar`

Lösn. uppg. 26. *Stränginterpolatorn s.*

a)

```
Namnet 'Kim Finkodare' har 12 bokstäver.
```

b)

```
println(s"$f har  ${f.size} bokstäver.")
println(s"$e har  ${e.size} bokstäver.")
```

Lösn. uppg. 27. *Tilldelningsoperatorer.*

Efter rad1: `a: Int` 40

Efter rad2: `a: Int` 40 `b: Int` 80

Efter rad3: `a: Int` 50 `b: Int` 80

Efter rad4: `a: Int` 50 `b: Int` 70

Efter rad5: `a: Int` 100 `b: Int` 70

Efter rad6: `a: Int` 100 `b: Int` 35

Lösn. uppg. 28. *Stora tal.*

a) `BigInt` kan användas i stället för `Int` vid mycket stora heltal. Det finns förståss även `Long` som har dubbelt omfång jämfört med `Int`, medan `BigInt` kan ha godtyckligt många siffror (ända tills minnet tar slut) och kan därmed representera ofantligt stora tal. `BigDecimal` kan användas i stället för `Double` vid mycket stora decimaltal.

b)

```
scala> BigInt(2).pow(64)
res0: scala.math.BigInt = 18446744073709551616
```

c) Beräkningar går mycket långsammare och de är lite krångligare att använda.

Lösn. uppg. 29. *Precedensregler*

a) `77:  Int`

b) `13: Int`

c) `-13: Int`

Lösn. uppg. 30. *Dokumentation av paket i Java och Scala.*

a) Scala: `Pi`, Java: `PI`

b) Man kan söka och filtrera fram alla förekomster av en viss teckenkombination.

c) Räknar ut hypotenusan (Pythagoras sats) utan risk för avrundningsproblem i mellanberäkningar.

Lösn. uppg. 31. *Noggrannhet och undantag i aritmetiska uttryck.*

a) `-2147483648` vilket motsvarar `Int.MinValue`.

b) Ett undantag kastas: `java.lang.ArithmeticException: / by zero`

c) `1.0000000000000001E8` (som förväntat)

d) Avrundas till `1E9` (flyttalsaritmetik med noggrannhetsproblem: ett stort flyttal plus ett (alltför) litet flyttal kan ge samma tal. Det lilla talet ”försvinner”).

e) `45.00000000000001` (flyttalsaritmetik med noggrannhetsproblem: enligt ”normal” aritmetik ska det bli exakt 45.)

f) `Infinity` (som även ges av `Double.PositiveInfinity` och som representerar den positiva oändligheten).

g) `2147483647` vilket motsvarar `Int.MaxValue`.

h) `NaN` vilket betyder ”Not a Number”.

i) `NaN` vilket betyder ”Not a Number”.

j) Ett undantag kastas: `java.lang.Exception: PANG!!!`

Lösn. uppg. 32. *Modulo-räkning med negativa tal.* I Scala har resultatet samma tecken som dividenden.

```
scala> 1 % 2
res0: Int = 1

scala> -1 % 2
res1: Int = -1

scala> -1 % -2
res2: Int = -1

scala> 1 % -2
res3: Int = 1
```

Lösn. uppg. 33. *Bokstavliga identifierare.*

a) Variabeln får namnet 'bokstavlig val', bakåt-apostrofer (eng. *backticks*) gör att man kan namnge variabler till annars otillåtna namn, t.ex. med mellanrum eller nyckelord i sig.

b) Backticks i Scala möjliggör alla möjliga tecken i namn. Exempel på användning: I java finns en metod som heter `java.lang.Thread.yield` men i Scala är yield ett nyckelord; för att komma runt det går det att i Scala skriva `java.lang.Thread.`yield``

Lösn. uppg. 34. *`java.lang.Integer`, hexadecimala litteraler, BigDecimal.*

a)

```
scala> import Integer.{toBinaryString => toBin, toHexString => toHex}

scala> for i <- Seq(33, 42, 64) do println(s"$i \t ${toBin(i)} \t ${toHex(i)}")
33   100001    21
42   101010    2a
64   1000000   40
```

b) Det hexadecimala heltalet $10c$ kan anges med litteralen `0x10c` i Scala, Java och många andra språk: [1]

```
scala> 0x10c
res0: Int = 268
```

c) [2]

```
scala> val c = 299792458
c: Int = 299792458

scala> BigDecimal(0x10).pow(c)
res68: scala.math.BigDecimal = 2.124892963227906613060986110887672E+360986089
```

Lösn. uppg. 35. *Strängformatering.*

```
val str = f"Jättegurkan är $g%1.3f meter lång"
```

(Om du tycker att `$g%1.3f` ser kryptiskt ut, så kan du trösta dig med att du nu får chansen att föra vidare ett anrikt arv från det urgamla språket C och den sägenomspunna funktionen `printf` till kommande generationer av invigda kodmagiker.)

Lösn. uppg. 36. *Multiplikationsvarning.*

a) Den andra multiplikationen flödar över (eng. *overflow*) gränsen för största möjliga värdet av en `Int`. I den tredje multiplikationen kastas i stället ett undantag `java.lang.ArithmeticException: integer overflow`

```
scala> Math.multiplyExact(1, 2)
res70: Int = 2

scala> Int.MaxValue * 2
res71: Int = -2

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

[1] https://en.wikipedia.org/wiki/0x10c
[2] https://c418.bandcamp.com/album/0x10c

```
java.lang.ArithmeticException: integer overflow
  at java.lang.Math.multiplyExact(Math.java:867)
  ... 42 elided
```

b) Används då man vill vara helt säker på att overflow-buggar ”smäller” direkt i stället för att generera felaktiga resultat vars konsekvenser kanske manifesterar sig långt senare. Dock är `multiplyExact` aningen långsammare än vanlig multiplikation.

Lösn. uppg. 37. *Extra operatorer för exakt multiplikation.*

a)

```
scala> Int.MaxValue *! 1
res0: Int = 2147483647

scala> Int.MaxValue *! 2
java.lang.ArithmeticException: integer overflow
  at java.lang.Math.multiplyExact(Math.java:867)
  at IntExtra.$times$bang(<console>:16)
  ... 32 elided
```

b)

```
extension (i: Int)
  def *!(j: Int) = Math.multiplyExact(i,j)
  def +!(j: Int) = Math.addExact(i,j)
  def -!(j: Int) = Math.subtractExact(i,j)
```

c) Det blir lätt väldigt kryptiskt med namn som består av flera specialtecken. Om du *verkligen* vill ha sådana operatorer är det *mycket* lämpligt att också erbjuda varianter i klartext:

```
extension (i: Int)
  def mulExact(j: Int) = Math.multiplyExact(i,j)
  def *!(j: Int) = i mulExact j

  def addExact(j: Int) = Math.addExact(i,j)
  def +!(j: Int) = i addExact j

  def subExact(j: Int) = Math.subtractExact(i,j)
  def -!(j: Int) = i subExact j
```

2. Lösning `programs`

2.1 Grunduppgifter

Lösn. uppg. 1. *Para ihop begrepp med beskrivning.*

kompilera	1	⇝	I	maskinkod skapas ur en eller flera källkodsfiler
skript	2	⇝	J	ensam kodfil, huvudprogram behövs ej
objekt	3	⇝	G	samlar variabler och funktioner
@main	4	⇝	L	där exekveringen av kompilerat program startar
programargument	5	⇝	A	kan överföras via parametern args till main
datastruktur	6	⇝	B	många olika element i en helhet; elementvis åtkomst
samling	7	⇝	C	datastruktur med element av samma typ
sekvenssamling	8	⇝	F	datastruktur med element i en viss ordning
Array	9	⇝	K	en förändringsbar, indexerbar sekvenssamling
Vector	10	⇝	E	en oföränderlig, indexerbar sekvenssamling
Range	11	⇝	N	en samling som representerar ett intervall av heltal
yield	12	⇝	O	används i for-uttryck för att skapa ny samling
map	13	⇝	H	applicerar en funktion på varje element i en samling
algoritm	14	⇝	M	stegvis beskrivning av en lösning på ett problem
implementation	15	⇝	D	en specifik realisering av en algoritm

Lösn. uppg. 2. *Använda terminalen.*

a)

```
> mkdir hello
> cd hello
> pwd
```

b)

```
> cd ..
> ls
```

Lösn. uppg. 3. *Skapa och köra ett Scala-skript.*

a)

```
Summan av de 1000 första talen är: 500500
```

b) Kompileringsfelet blir: `')' expected, but eof found`

c) Filen ska se ut så här:

```
val n = args(0).toInt
val summa = (1 to n).sum
println(s"Summan av de $n första talen är: $summa")
```

Utskriften blir så här:

```
Summan av de 5001 första talen är: 12507501
```

d) Körtidsfelet blir:

```
java.lang.ArrayIndexOutOfBoundsException: Index 0 out of bounds for length 0
```

Eftersom rrayen `args` är tom om programargument saknas så finns ej platsen med index 0.

Lösn. uppg. 4. *Scala-applikation med* `@main`.

a) Kompilatorn har skapat 5 filer i underkataloger till `.scala-build` som heter:

```
'hello$package.class'  'hello$package$.class'  'hello$package.tasty'
run.class   run.tasty
```

b) Felmeddelandet får du om du tar bort den sista krullparentesen. `eof` i felmeddelandet står för end-of-file. Detta felmeddelande är vanligt vid oparade parenteser, men kompilatorn har ofta extra svårt att ge bra felmeddelande om en av parenteserna i ett parentespar saknas och det kan hända att den pekar ut felaktig rad för positionen där det som saknas borde stå.

c) `Syntax Error: Expected a toplevel definition`. Utan klammerparenteser så är det indenteringarna som bestämmer vilka delar av koden som hör samman. Om du tar bort indenteringen på den sista raden med utskrift-satsen så tolkar kompilatorn detta som att denna ligger *utanför* main-funktionen och du får ett felmeddelande eftersom det inte är tillåtet att ha ensamma satser på toppnivå. (Det går dock bra att ha ensamma satser i ett skript med `.sc` i slutet av namnet på kodfilen.)

d) Annoteringen `@main` berättar för kompilatorn att funktionen är ett huvudprogram kan utgöra en startpunkt för exekveringen.

Under huven skapar kompilatorn ett objekt med samma namn som ditt huvudprogram. I det objektet genererar kompilatorn i sin tur en metod med namnet `main` som tar en sträng-array som parameter och har returtypen `Unit`. Ett kompilerat program måste ha minst ett objekt med exakt en sådan main-metod eftersom exekveringsmiljön JVM förutsätter detta och anropar en sådan main-metoden med en sträng-array innehållande eventuella programargument när exekveringen startar.

Ett alternativ till `@main` är att definiera en s.k. *primitiv* main-metod i ett singelobjekt. (Detta är nödvändigt i gamla Scala 2, innan den enklare `@main`.annoteringen kom i Scala 3.)

```
object Hello:
  def main(args: Array[String]): Unit =
    val message = "Hello world!"
    println(message)
```

Lösn. uppg. 5. *Skapa och använda samlingar.*

`val xs = Vector(2)`	1	⇝ I	ny referens till sekvens av längd 1
`val ys = Array.fill(9)(0)`	2	⇝ C	ny referens till förändringsbar sekvens
`Vector.fill(9)(' ')`	3	⇝ J	ny oföränderlig sekvens med blanktecken
`xs(0)`	4	⇝ E	förkortad skrivning av `apply(0)`
`xs.apply(0)`	5	⇝ F	indexering, ger första elementet
`xs :+ 0`	6	⇝ A	ny samling med en nolla tillagd på slutet
`0 +: xs`	7	⇝ H	ny samling med en nolla tillagd i början
`ys.mkString`	8	⇝ K	ny sträng med alla element intill varandra
`ys.mkString(",")`	9	⇝ G	ny sträng med komma mellan elementen
`xs.map(_.toString)`	10	⇝ D	ny samling, elementen omgjorda till strängar
`xs.map(_.toInt)`	11	⇝ B	ny samling, elementen omgjorda till heltal

Lösn. uppg. 6. *Jämför `Array` och `Vector`.*

a)

Vector	1	⇝ B	oföränderlig
Array	2	⇝ A	förändringsbar

b)

Vector	1	⇝ B	varianter med fler/andra element skapas snabbt ur befintlig
Array	2	⇝ A	långsam vid ändring av storlek (kopiering av rubbet krävs)

c)

Vector	1	⇝ A	`xs == ys` är `true` om alla element lika
Array	2	⇝ B	olikt andra Scala-samlingar kollar == ej innehållslikhet

Lösn. uppg. 7. *Räkna ut summa, min och max i `args`.*

```
@main def sumMinMax(args: Int*): Unit =
  println(s"${args.sum} ${args.min} ${args.max}")
```

```
> scala-cli run sum-min-max.scala -- hej
Illegal command line: java.lang.NumberFormatException: For input string: "hej"
```

Lösn. uppg. 8. *Algoritm: SWAP.*

a) Pseudokoden kan se ut såhär:

```
Deklarera heltalsvariabel temp.
Kopiera värdet från x till temp.
Kopiera värdet från y till x.
```

```
Kopiera värdet från temp till y.
```

b)

Du behöver deklarera en temporär variabel där du kan spara undan ett av värdena, så det inte skrivs över vid första tilldelningen.

```
val temp = x
x = y
y = temp
```

Lösn. uppg. 9. *Indexering och tilldelning i Array med SWAP.*

```
@main def swapFirstLastArg(args: String*): Unit =
  val xs = args.toArray
  if xs.length > 1 then
    val temp = xs(0)
    xs(0) = xs(xs.length - 1)
    xs(xs.length - 1) = temp
  println(xs.mkString(" "))
```

Lösn. uppg. 10. ***for****-uttryck och* `map`*-uttryck.*

```
for x <- xs yield x * 2
for i <- xs.indices yield i
xs.map(x => x + 1)
for i <- 0 to 1 yield xs(i)
(1 to 3).map(i => i)
(1 until 3).map(i => xs(i))
```

1	⇝	A	`Vector(2, 4, 6)`
2	⇝	E	`Vector(0, 1, 2)`
3	⇝	D	`Vector(2, 3, 4)`
4	⇝	B	`Vector(1, 2)`
5	⇝	C	`Vector(1, 2, 3)`
6	⇝	F	`Vector(2, 3)`

Lösn. uppg. 11. *Algoritm: SUMBUG*

a) Bugg: Eftersom `i` inte inkrementeras, fastnar programmet i en oändlig loop. Fix: Lägg till en sats i slutet av while-blocket som ökar värdet på i med 1. Bugg: Eftersom man bara ökar summan med 1 varje gång, kommer resultatet att bli summan av n stycken 1or, inte de n första heltalen. Fix: Ändra så att summan ökar med `i` varje gång, istället för 1. För -1, blir resultatet 0. Förklaring: i börjar på 1 och är alltså aldrig mindre än n som ju är -1. while-blocket genomförs alltså noll gånger, och efter att `sum` får sitt ursprungsvärde förändras den aldrig.

b) Summan blir 39502716.
Såhär kan en implementation se ut:

```
@main def sumn(n: Int): Unit =
  var sum = 0
  var i = 1
  while i <= n do
    sum = sum + i
    i = i + 1
  println(sum)
```

2.2 Extrauppgifter; träna mer

Lösn. uppg. 12. *Algoritm: MAXBUG*

a) Bugg: `i` inkrementeras aldrig. Programmet fastnar i en oändlig loop. Fix: Lägg till en sats som ökar i med 1, i slutet av while-blocket.

b) Så här kan implementationen se ut:

```
@main def maxn(args: String*): Unit =
  var max = Int.MinValue
  val n = args.length
  var i = 0
  while i < n do
    val x = args(i).toInt
    if x > max then
      max = x
    i += 1
  println(max)
```

c) Raden där max initieras ändras till `var max = args(0).toInt`

d) För att inte få `java.lang.IndexOutOfBoundsException: 0` behövs en kontroll som säkerställer att inget görs om samlingen `args` är tom:

```
@main def maxn(args: String*): Unit =
  if args.size > 0 then
    var max = args(0).toInt
    val n = args.size
    var i = 0
    while i < n do
      val x = args(i).toInt
      if x > max then
        max = x
      i += 1
    println(max)
  else
    println("Empty")
```

Lösn. uppg. 13. *Algoritm MIN-INDEX.*

a) En onödig jämförelse sker, men resultatet påverkas ej.

b)

```
def indexOfMin(xs: Array[Int]): Int =
  var minPos = 0
  var i = 1
  while i < xs.size do
    if xs(i) < xs(minPos) then
      minPos = i
    i += 1
  if xs.size > 0 then minPos else -1
```

Lösn. uppg. 14. *Datastrukturen* `Range`.

a) värde: `Range(1,2,3,4,5,6,7,8,9)`
typ: `scala.collection.immutable.Range`

b) värde: `Range(1,2,3,4,5,6,7,8,9,10)`
typ: `scala.collection.immutable.Range`

c) värde: `Range(0,5,10,15,20,25,30,35,40,45)`
typ: `scala.collection.immutable.Range`

d) värde: `10`, typ: `Int`

e) värde: `Range(0,5,10,15,20,25,30,35,40,45,50)`
typ: `scala.collection.immutable.Range`

f) värde: `11`, typ: `Int`

g) värde: `Range(0,1,2,3,4,5,6,7,8,9)`
typ: `scala.collection.immutable.Range`

h) värde: `Range(0,1,2,3,4,5,6,7,8,9)`
typ: `scala.collection.immutable.Range`

i) värde: `Range(0,1,2,3,4,5,6,7,8,9)`
typ: `scala.collection.immutable.Range`

j) värde: `Range(0,1,2,3,4,5,6,7,8,9,10)`
typ: `scala.collection.immutable.Range.Inclusive`

k) värde: `Range(0,1,2,3,4,5,6,7,8,9,10)`
typ: `scala.collection.immutable.Range.Inclusive`

l) värde: `Range(0,5,10,15,20,25,30,35,40,45)`
typ: `scala.collection.immutable.Range`

m) värde: `Range(0,5,10,15,20,25,30,35,40,45,50)`
typ: `scala.collection.immutable.Range`

n) värde: `11`, typ: `Int`

o) värde: `500500`, typ: `Int`

2.3 Fördjupningsuppgifter; utmaningar

Lösn. uppg. 15. *Sten-Sax-Påse-spel.* En (lättbegriplig?) lösning som provar alla kombinationer:

```
def winner(user: Int, computer: Int): String =
  if      choices(user) == "Sten" && choices(computer) == "Påse" then "Datorn"
  else if choices(user) == "Sten" && choices(computer) == "Sax"  then "Du"
  else if choices(user) == "Påse" && choices(computer) == "Sten" then "Du"
  else if choices(user) == "Påse" && choices(computer) == "Sax"  then "Datorn"
  else if choices(user) == "Sax"  && choices(computer) == "Sten" then "Datorn"
  else if choices(user) == "Sax"  && choices(computer) == "Påse" then "Du"
  else "Ingen"
```

En klurigare lösning (och svårbegripligare?) med hjälp av modulo-räkning:

```
def winner(user: Int, computer: Int): String =
  val result = (user - computer + 3) % 3
  if user == computer then "Ingen"
  else if result == 1 then "Du"
  else "Datorn"
```

Moduloräkningen kräver att elementen i `choices` är i *förlorar-över*-ordning, alltså Sten, Påse, Sax. Addition med 3 görs för att undvika negativa tal, som beter sig annorlunda i moduloräkning.

Lösn. uppg. 16. *Jämför exekveringstiden för storleksförändring mellan `Array` och `Vector`.*

a) Med en dator som har en `i7-4790K CPU @ 4.00GHz` blev det så här:

```
scala> def time(block: => Unit): Double =
     |   val t = System.nanoTime
     |   block
     |   (System.nanoTime - t)/1e6  // ger millisekunder
def time(block: => Unit): Double

scala> val as = Array.fill(1e6.toInt)(0)
val as: Array[Int] = Array(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
large output truncated, print value to show all

scala> val vs = Vector.fill(1e6.toInt)(0)
val vs: Vector[Int] = Vector(0, 0, 0, 0, 0, ...
large output truncated, print value to show all

scala> val ast = (for i <- 1 to 10 yield time(as :+ 0)).sum / 10.0
val ast: Double = 1.8719819999999998

scala> val vst = (for i <- 1 to 10 yield time(vs :+ 0)).sum / 10.0
val vst: Double = 0.006485099999999999

scala> ast / vst
val res3: Double = 288.6589258453995
```

b) `Vector` är två tiopotenser snabbare i detta exempel. Anledningen är att varje storleksförändring av en `Array` kräver allokering och elementvis kopiering av en helt

ny `Array` medan den oföränderliga `Vector` kan återanvända hela datastrukturen med redan allokerade element när nya element läggs till.

Lösn. uppg. 17. *Minnesåtgång för* `Range`.

a) Variabeln `intervall` refererar till objekt som tar upp 12 bytes.

b) Variabeln `sekvens` refererar till objekt som tar upp ca 4 miljarder bytes.

Lösn. uppg. 18. *Undersök den genererade byte-koden.*

a) Så här ser funktionen `plusxy` ut:

```
1  public int plusxy(int, int);
2    descriptor: (II)I
3    flags: (0x0001) ACC_PUBLIC
4    Code:
5      stack=2, locals=3, args_size=3
6         0: iload_1
7         1: iload_2
8         2: iadd
9         3: ireturn
10     LineNumberTable:
11       line 2: 0
12     LocalVariableTable:
13       Start  Length  Slot  Name   Signature
14           0       4     0  this   Lplusxy$package$;
15           0       4     1     x   I
16           0       4     2     y   I
```

Det är instruktionen `iadd` som gör själva additionen.

b) Det har tillkommit en parameter till i byte-koden. Instruktionen `iadd` görs nu två gånger. Instruktionen `iadd` adderar exakt två tal i taget.

```
1  public int plusxyz(int, int, int);
2    descriptor: (III)I
3    flags: (0x0001) ACC_PUBLIC
4    Code:
5      stack=2, locals=4, args_size=4
6         0: iload_1
7         1: iload_2
8         2: iadd
9         3: iload_3
10        4: iadd
11        5: ireturn
12     LineNumberTable:
13       line 2: 0
14     LocalVariableTable:
15       Start  Length  Slot  Name   Signature
16           0       6     0  this   Lplusxyz$package$;
17           0       6     1     x   I
18           0       6     2     y   I
19           0       6     3     z   I
```

c) Prefixet `i` i instruktionsnamnet `iadd` står för ”integer” och anger att heltalsdivision avses.

3. Lösning `functions`

Lösn. uppg. 1. *Para ihop begrepp med beskrivning.*

funktionshuvud	1	⇝	D	har parameterlista och eventuellt en returtyp
funktionskropp	2	⇝	G	koden som exekveras vid funktionsanrop
parameterlista	3	⇝	I	beskriver namn och typ på parametrar
block	4	⇝	M	kan ha lokala namn; sista raden ger värdet
namngivna argument	5	⇝	N	gör att argument kan ges i valfri ordning
defaultargument	6	⇝	A	gör att argument kan utelämnas
värdeanrop	7	⇝	L	argumentet evalueras innan anrop
namnanrop	8	⇝	E	fördröjd evaluering av argument
äkta funktion	9	⇝	C	ger alltid samma resultat om samma argument
predikat	10	⇝	J	en funktion som ger ett booleskt värde
slumptalsfrö	11	⇝	F	ger återupprepningsbar sekvens av pseudoslumptal
anonym funktion	12	⇝	B	funktion utan namn; kallas även lambda
rekursiv funktion	13	⇝	K	en funktion som anropar sig själv
stack trace	14	⇝	H	lista anropskedja vid körtidsfel

Lösn. uppg. 2. *Definiera och anropa funktioner.*

a)

```
def öka(x: Int = 1): Int = x + 1
```

b) `5`

c)

```
def minska(x: Int = 1): Int = x - 1
```

d) `1`

e)

- *Kort, förenklad förklaring:* Parametern i funktionshuvudet är ett lokalt namn på indata som kan användas i funktionskroppen, medan argumentet är själva värdet på parametern som skickas med vid anrop.
- *Längre, mer exakt förklaring:* En **parameter** är en deklaration av en oföränderlig variabel i ett funktionshuvud vars namn finns tillgängligt lokalt i funktionskroppen. Vid anrop *binds* parameternamnet till ett specifikt argument. Ett **argument** är ett uttryck som appliceras på en funktion vid anrop. Normalt evalueras argumentet innan anropet sker, men om parametertypen föregås av `=>` fördröjs evalueringen av argumentet och sker i stället *varje gång* parameternamnet förekommer i funktionskroppen.

Lösn. uppg. 3. *Textspelet AliensOnEarth.*

a) `"penguin"` är bästa alternativ med sannolikheten $\frac{1}{2} + \frac{1}{2} \cdot \frac{1}{3} = \frac{2}{3}$

b)

`options.indices`	1	⇝	B	heltalssekvens med alla index i en sekvens
`"1X2".toLowercase`	2	⇝	A	gör om en sträng till små bokstäver
`Random.nextInt(n)`	3	⇝	C	slumptal i intervallet `0 until n`
`try { } catch { }`	4	⇝	E	fångar undantag för att förhindra krasch
`""" ... """`	5	⇝	F	sträng som kan sträcka sig över flera kodrader
`s.stripMargin`	6	⇝	D	tar bort marginal till och med vertikalstreck
`e.printStackTrace`	7	⇝	G	skriver ut information om ett undantag

Lösn. uppg. 4. *Äkta funktioner.*

- Funktionerna `inc`, `addY` och `isPalindrome` är äkta. Notera att `y`-variablen initialiseras till `0` och kan sedan inte ändras eftersom den är deklarerad med nyckelordet `val`.

Lösn. uppg. 5. *Applicera funktion på varje element i en samling. Funktion som argument.*

`for i <- 1 to 3 yield öka(i)`	1	⇝	E	`Vector(2, 3, 4)`
`Vector(2, 3, 4).map(i => öka(i))`	2	⇝	A	`xs`
`xs.map(öka)`	3	⇝	B	`Vector(4, 5, 6)`
`xs.map(öka).map(öka)`	4	⇝	D	`Vector(5, 6, 7)`
`xs.foreach(öka)`	5	⇝	C	`()`

Lösn. uppg. 6. *Funktion som äkta värde.*

a)

`fleraAnrop(1, hälsa)`	1	⇝	D	`f2("Hej!")`
`fleraAnrop(3, hälsa)`	2	⇝	B	`fleraAnrop(3, f1)`
`fleraAnrop(2, f1)`	3	⇝	A	`f2("Hej!\nHej!")`
`fleraAnrop(1, f3)`	4	⇝	C	`f3()`

b) `f1` och `f3` är av typen `() => Unit` och `f2` av typen `String => Unit`.

c) Nej. `f1` och `f2` är av två olika funktionstyper.

d) Ja, det går fint.

e) Nej. När funktionen inte har någon parameter behöver kompilatorn mer information för att vara säker på att det är ett funktionsvärde du vill ha.

f) Ja! Nu med typinformationen på plats är kompilatorn säker på vad du vill göra.

Lösn. uppg. 7. *Anonyma funktioner.*

`(0 to 2).map(i => i + 1)`	1	⇝ B	`(2 to 4).map(i => i - 1)`
`(1 to 3).map(_ + 1)`	2	⇝ D	`Vector(2, 3, 4)`
`(2 to 4).map(math.pow(2, _))`	3	⇝ E	`Vector(4.0, 8.0, 16.0)`
`(3 to 5).map(math.pow(_, 2))`	4	⇝ A	`Vector(9.0, 16.0, 25.0)`
`(4 to 6).map(_.toDouble).map(_ / 2)`	5	⇝ C	`Vector(2.0, 2.5, 3.0)`

Lösn. uppg. 8. *Lär dig läsa en stack trace.*En stack trace innehåller följande information:

1. ett felmeddelande
2. namn på alla funktioner som anropats vid tiden för körtidsfelet, enligt alla aktiveringsposter som ligger på anropsstacken
3. aktuell namnrymnd för varje funktionen, alltså paket/singelobjekt
4. namnet på kodfilen för varje funktion
5. radnummer i varje funktion
6. den funktion som kommer först är den funktion där felet inträffade
7. eventuellt kan felet inträffa i standardbibliotekets funktioner och då är din egen funktion tidigare i anropskedjan

Exempel på en stack trace:

```
> cat fel.scala
@main def run =
  println("Hej Scala!" + Vector().head)
> scala-cli run fel.scala
Compiling project (Scala 3.3.0, JVM)
Compiled project (Scala 3.3.0, JVM)
Exception in thread "main" java.util.NoSuchElementException: empty.head
  at scala.collection.immutable.Vector.head(Vector.scala:279)
  at fel$package$.run(fel.scala:2)
  at run.main(fel.scala:1)
>
```

3.1 Extrauppgifter; träna mer

Lösn. uppg. 9. *Funktion med flera parametrar.*

a) `-100`

b) `15`

c) `185`

d) `256`

Lösn. uppg. 10. *Medelvärde.*

```
def avg(x: Int, y: Int): Double = (x + y) / 2.0
```

Lösn. uppg. 11. *Funktionsanrop med namngivna argument.*

a)

```
Namn: Triangelsson, Stina
Namn: Oval, Viktor
```

b)

- Anroparen kan själv välja ordning.
- Koden blir lättare att begripa om parameternamnen är självbeskrivande.
- Hjälper till att förhindra buggar som beror på förväxlade parametrar.

Lösn. uppg. 12. *Bortkastade resultatvärden och returtypen* `Unit`.

a) Procedurer returnerar tomma värdet och `println` är en procedur. När tomma värdet skrivs ut visas `()`.

b) Procedurer returnerar tomma värdet. Om du anger returtyp `Unit` explicit, har du bättre chans att kompilatorn kan ge varning då uträkningar kommer att kastas bort. En varning avbryter inte exekveringen, utan är ett sätt för kompilatorn att ge dig tips om saker som kan behöva fixas till i din kod.

c) I Scala är variabeldeklaration, precis som en tilldelningssats, och inte ett uttryck och saknar värde.

d) Koden blir lättare att läsa och kompilatorn får bättre möjlighet att hjälpa till med varningar om resultatvärden riskerar att bli bortkastade.

3.2 Fördjupningsuppgifter; utmaningar

Lösn. uppg. 13. *Föränderlighet av parametrar.*

a) Nej, i Scala är parametern oföränderlig och det blir kompileringsfel om man försöker tilldela den ett nytt värde i funktionskroppen.

b) c) Ja det går utmärkt i både Java och Python att ändra värdet på parametern i funktionskroppen med tilldelning, men koden riskerar att bli förvirrande.
https://stackoverflow.com/questions/2970984

Lösn. uppg. 14. *Värdeanrop och namnanrop.*

a) Vid varje anrop av `snark` sker en utskrift och en fördröjnig innan 42 returneras. `42 + 42 == 84` vilket blir värdet av uttrycket.

```
scala> snark + snark
snark snark val res1: Int = 84
```

b) Uttrycket `snark` evalueras direkt vid anropet och parametern `x` binds till värdet 42 och i funktionskroppen beräknas $42 + 42$. Utskriften sker bara en gång.

```
callByValue(snark)
snark val res2: Int = 84
```

c) Evalueringen av uttrycket `snark` fördröjs tills varje förekomst av parametern `x` i funktionskroppen. Utskriften sker två gånger.

```
callByName(snark)
snark snark val res3: Int = 84
```

d) Evalueringen av uttrycket `zzz` fördröjs tills varje förekomst av parametern `x` i funktionskroppen. Utskriften sker en gång eftersom **`val`**-variabler tilldelas sitt värde en gång för alla vid den fördröjda initialiseringen.

```
callByName(zzz)
snark val res4: Int = 84
```

Lösn. uppg. 15. *Skapa din egen kontrollstruktur med hjälp av namnanrop.*

a) Blocket är ett uttryck som har värdet `(): Unit`. Evalueringen av blocket sker där namnet `b` förekommer i procedurkroppen, vilket är två gånger.

```
scala> görDettaTvåGånger { println("goddag") }
goddag
goddag
```

b)

```
def upprepa(n: Int)(block: => Unit): Unit =
  var i = 0
  while i < n do
    block
    i += 1
```

c)

```
upprepa(100):
  val tärningskast = (math.random() * 6 + 1).toInt
  print(s"\$tärningskast ")
```

d)

```
def repeat(n: Int)(p: Int => Unit): Unit =
  var i = 0
  while i < n do
    p(i)
    i += 1
  end while
end repeat
```

e)

```
repeat(100){ i =>
  print(s"$i: ")
  println(math.random())
}
```

Du kan använda färre klammerparenteser med hjälp av kolon:

```
repeat(100): i =>
  print(s"$i: ")
  println(math.random())
```

Lösn. uppg. 16. *Uppdelad parameterlista och stegade funktioner.*

a)

```
1 scala> def add2(a: Int)(b: Int) = a + b
2 def add2(a: Int)(b: Int): Int
3
4 scala> add2(1)(1)
5 val res0: Int = 2
```

b)

- Rad 3:

  ```
  doremi doremi doremi
  ```

- Rad 5:

  ```
  lalalalalalala
  ```

Lösn. uppg. 17. *Rekursion.*

a) `countdown` skriver ut `x` och gör ett rekursivt anrop med `x - 1` som argument, men bara om basvillkoret `x > 0` är uppfyllt. Resultatet blir en ändlig repetition. `finalCountdown` anropar sig själv rekursivt men saknar ett basvillkor som kan avbryta rekursionen, vilket genererar en oändlig repetition. Vid -128 blir det *overflow* eftersom bitarna inte räcker till för större negativa tal och räkningen börjar om på 127. (Om minskar fördröjningen till `Thread.sleep(1)` blir det ganska snabbt *stack overflow*)

b) Eftersom vi hade `1/x` *efter* det rekursiva anropet i föregående deluppgift, så kom vi aldrig till denna (potentiellt ödesdigra) beräkning, utan lade bara aktiveringsposter på hög på stacken vid varje anrop. Om vi placerar `1/x` *före* det rekursiva anropet, så når vi detta uttryck direkt och det kastas ett undantag p.g.a. division med noll.

c) Den sista raden leder till många fler rekursiva anrop, så som basvillkoret och det rekursiva anropet är konstruerade. Lägg gärna in en `println`-sats före det rekursiva anropet och undersök i detalj vad som sker.

Lösn. uppg. 18. *Undersök svansrekursion genom att kasta undantag.* `countdown` är svansrekursiv eftersom det rekursiva anropet står *sist* och kan då optimeras till en `while`-loop av kompilatorn. Det går fint att köra ända till det exploderar, även med 10000 anrop, och i felmeddelandet finns det endast ett anrop till `countdown`.

`countdown2` är inte svansrekursiv eftersom den har ett uttryck `efter` det rekursiva anropet. I felutskriften syns alla rekursiva anrop till `countdown2` innan basvillkoret inträffade. Vid `countdown2(10000)` uppfylls inte basvillkoret innan det blir `StackOverflowError`.

Lösn. uppg. 19. *`@tailrec`-annotering.* Första gången `countNoTailrec(100000L)` anropas blir det `StackOverflowError`. Med annoteringen `@tailrec` får vi ett kompileringsfel eftersom kompilatorn inte kan optimera en icke svansrekursiv funktion. Om funktionen skrivs om kan kompilatorn optimera funktionen så att rekursionen byts ut mot en `while`-loop och vi kan köra så länge vi orkar utan att stacken flödar över. Och himla snabbt går det!!

4. Lösning objects

4.1 Grunduppgifter; förberedelse inför laboration

Lösn. uppg. 1. *Para ihop begrepp med beskrivning.*

modul	1	⇝	C	kodenhet med abstraktioner som kan återanvändas
singelobjekt	2	⇝	B	modul som kan ha tillstånd; finns i en enda upplaga
paket	3	⇝	D	modul som skapar namnrymd; maskinkod får egen katalog
import	4	⇝	F	gör namn tillgängligt lokalt utan att hela sökvägen behövs
export	5	⇝	P	gör namn synligt utåt som medlem i detta objekt
lat initialisering	6	⇝	G	allokering sker först när namnet refereras
medlem	7	⇝	E	tillhör ett objekt; nås med punktnotation om synlig
attribut	8	⇝	H	variabel som utgör (del av) ett objekts tillstånd
metod	9	⇝	A	funktion som är medlem av ett objekt
privat	10	⇝	K	modifierar synligheten av en objektmedlem
överlagring	11	⇝	J	metoder med samma namn men olika parametertyper
namnskuggning	12	⇝	L	lokalt namn döljer samma namn i omgivande block
namnrymd	13	⇝	I	omgivning där är alla namn är unika
enhetlig access	14	⇝	M	ändring mellan def och val påverkar ej användning
punktnotation	15	⇝	O	används för att komma åt icke-privata delar
typalias	16	⇝	N	alternativt namn på typ som ofta ökar läsbarheten

Lösn. uppg. 2. *Nästlade singelobjekt, import, synlighet och punktnotation.*

a)

```
object Underjorden:
  var x = 0
  var y = 1

  object Mullvaden:
    var x = Underjorden.x + 10
    var y = Underjorden.y + 9

  object Masken:
    private var x = Mullvaden.x
    var y = Mullvaden.y + 190
    def ärMullvadsmat: Boolean = x == Mullvaden.x && y == Mullvaden.y
```

b)

```
scala> :load Underjorden.scala
scala> import Underjorden.*
scala> Masken.ärMullvadsmat
val res0: Boolean = false
scala> Masken.y = Mullvaden.y
scala> Masken.ärMullvadsmat
```

```
val res1: Boolean = true
```

c)

```
scala> import Mullvaden.*
scala> import Masken.*
scala> x = -1
scala> Mullvaden.x
val res2: Int = -1

scala> Masken.x
1 |Masken.x
  |^^^^^^^^
  |variable x cannot be accessed as a member of Underjorden.Masken.type from mo

scala> Underjorden.x
val res3: Int = 0
```

Förklaring: När importen av Maskens alla synliga medlemmar sker kommer de som ej är privata att överskugga andra medlemmar med samma namn. Det är Mullvadens `x`-variabel som tilldelas `-1` eftersom Maskens `x` är privat och ej syns utåt. Underjordens medlemmar blir överskuggade av Maskens `y` och Mullvadens `x` men man kan komma åt dem genom att använda punktnotation.

Lösn. uppg. 3. *Export.*

a) Likhet: Både **`import`** och **`export`** styr synlighet. Skillnad: **`import`** styr lokal synlighet *inuti* ett objekt medan **`export`** styr synlighet *utanför* ett objekt.

b) Man kan med **`export`** på ett smidigt sätt plocka ihop medlemmar från andra objekt och göra dem synliga från mitt eget objekt.

```
object MittObjekt:
  export java.awt.Color.*  // alla färger blir medlemmar i MittObjekt
  export math.{atan2, Pi}  // atan2 och Pi blir medlemmar i MittObjekt
```

```
scala> object MittObjekt:
     |   export java.awt.Color.*
     |   export math.{atan2, Pi}
     |

scala> MittObjekt.RED
val res0: java.awt.Color = java.awt.Color[r=255,g=0,b=0]

scala> MittObjekt.atan2(3,3) / MittObjekt.Pi
val res1: Double = 0.25
```

Lösn. uppg. 4. *Tupler.*

a) `djup` har typen `Double`.

b) `hemlis` har typen `(String, (Int, Int, Double))`.

c)

```
object Underjorden3D:
  private val hemlis = ("uppgången till överjorden", (3, 4, 0.0))
```

```
object Mullvaden:
  var pos = (5, 3, math.random() * 10 + 1)

  def djup: Double  = pos._3

object Masken:
  private var pos = (0, 0, 10.0)

  def ärMullvadsmat: Boolean = pos == Mullvaden.pos

  def ärRaktUnderUppgången: Boolean =
    pos._1 == hemlis._2._1 && pos._2 == hemlis._2._2
```

d) Noll-tupeln.

Lösn. uppg. 5. *Lat initialisering.*

a) `"nu!"` skrivs bara ut första gången z används.

```
scala> z
nu!
val res19: Array[Int] = Array(0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

scala> z
val res20: Array[Int] = Array(0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

b) Allokeringen av arrayen sker första gången z används (och inte vid deklarationen).

```
scala> lazy val z = { println("nu!"); Array.fill(1e9.toInt)(0)}
val z: Array[Int] = <lazy>

scala> z
nu!
java.lang.OutOfMemoryError: Java heap space
```

c) Utskriften av `"nu!"` sker först när singelobjektet `zzz` används för första gången. Vi borde lägga initialiseringen av b före a eller göra a till en **`lazy val`**.

d)

```
scala> import test.*
import test.*

scala> zzz.a      // först när vi använder zzz skrivs "nu!"
nu!               // detta skedde *inte* när vi importerade test
val res0: Int = 42

scala> buggig.a   // a blir 0 eftersom b inte är initialiserad
val res1: Int = 0

scala> funkar.a   // med lazy val unviker vi problemet
val res2: Int = 42


scala> zzz.a     // andra gången är init redan gjort och ingen "nu!"
val res3: Int = 42
```

e) `lazy val a = uttryck` innebär att initialiseringsuttrycket evalueras *en* gång, men evalueringen skjuts på framtiden tills det eventuellt händer att namnet `a` används, medan `def b = uttryck` innebär att funktionskroppens uttryck evalueras *varje gång* namnet `b` (eventuellt) används.

Lösn. uppg. 6. *Extensionsmetoder.*

a)

```
scala> extension (i: Int) def inc = i + 1
```

b)

```
scala> extension (i: Int) def dec = i - 1
```

c)

```
extension (i: Int)
  def inc = math.incrementExact(i)
  def dec = math.decrementExact(i)
```

d) Med `math.incrementExact` och `math.decrementExact` ges exception om vi går över gränsen:

```
scala> math.incrementExact(Int.MaxValue)
java.lang.ArithmeticException: integer overflow
  at java.base/java.lang.Math.incrementExact(Math.java:1023)
  at scala.math.incrementExact(package.scala:418)
  ... 34 elided
```

Lösn. uppg. 7. *Extensionsmetoder.*

a) Enligt dokumentationen har `PixelWindow`-klassen dessa parametrar:

- `width : Int` anger fönstrets bredd, defaultargument `800`
- `height: Int` anger fönstrets höjd, defaultargument `640`
- `title : String` anger fönstrets titel, defaultargument `"PixelWindow"`
- `background: Color` anger bakgrundsfärg, defaultargument `java.awt.Color.black`
- `foreground: Color` anger bakgrundsfärg, defaultargument `java.awt.Color.green`

Man kan skapa nya fönsterinstanser till exempel så här:

```
val w1 = new introprog.PixelWindow()
val w2 = new introprog.PixelWindow(100, 200, "Mitt fina nya fönster")
```

b) Du kan även ladda ner senaste `introprog` så här:

```
curl -o introprog_3-1.3.1.jar -sLO https://fileadmin.cs.lth.se/introprog.jar
```

c)

```
1 > scala repl --jar introprog_3-1.3.1.jar
2 scala> val w = new introprog.PixelWindow(400,300,"HEJ")
3 scala> w.line(100, 100, 200, 100)
4 scala> w.line(200, 100, 200, 200)
5 scala> w.line(200, 200, 100, 200)
6 scala> w.line(100, 200, 100, 100)
```

d)

```
package hello

object Main:
  val w = new introprog.PixelWindow(400, 300, "HEJ")

  var color = java.awt.Color.red

  def square(p: (Int, Int))(side: Int): Unit =
    if side > 0 then
      // side == 1 ger en kvadrat som är en enda pixel
      val d = side - 1

      w.line(p._1,     p._2,     p._1 + d, p._2,     color)
      w.line(p._1 + d, p._2,     p._1 + d, p._2 + d, color)
      w.line(p._1 + d, p._2 + d, p._1,     p._2 + d, color)
      w.line(p._1,     p._2 + d, p._1,     p._2,     color)

  def main(args: Array[String]): Unit =
    println("Rita kvadrat:")
    square(300,100)(50)
```

e)

```
> scala-cli run hello-window.scala --jar introprog_3-1.3.1.jar --main-class hel
```

f)

```
> scala-cli run hello-window.scala --dep se.lth.cs::introprog:1.3.1 --main-clas
```

g)

```
//> using scala 3.3
//> using lib se.lth.cs::introprog:1.3.1
```

Lösn. uppg. 8. *Färg.*

a)

```
object Color:
  import java.awt.{Color as JColor}

  val mole   = new JColor( 51,  51,   0)
  val soil   = new JColor(153, 102,  51)
  val tunnel = new JColor(204, 153, 102)
```

b)

```
package hello

object Color:
  import java.awt.{Color as JColor}
```

```
  val mole   = new JColor( 51, 51,    0)
  val soil   = new JColor(153, 102, 51)
  val tunnel = new JColor(204, 153, 102)


object Main:
  val w = new introprog.PixelWindow(width = 400, height = 300, title = "HEJ")

  type Pt = (Int, Int)

  var color = java.awt.Color.red

  def rak(p:  Pt)(d: Int) = w.line(p._1, p._2, p._1 + d - 1, p._2, color)

  def fyll(p: Pt)(s: Int) = for i <- 0 until s do rak((p._1, p._2 + i))(s)

  def square(p: (Int, Int))(side: Int): Unit =
    if (side > 0) then
      val d = side - 1  // side == 1 ska ge en kvadrat som är en pixel stor
      w.line(p._1,     p._2,     p._1 + d, p._2,     color)
      w.line(p._1 + d, p._2,     p._1 + d, p._2 + d, color)
      w.line(p._1 + d, p._2 + d, p._1,     p._2 + d, color)
      w.line(p._1,     p._2 + d, p._1,     p._2,     color)

  def main(args: Array[String]): Unit =
    import Color.*
    color = soil
    fyll(100,100)(75)
    color = tunnel
    fyll(100,100)(50)
    color = mole
    fyll(150,150)(25)
```

c) Vid anropen av `rak` och `fyll` utnyttjas att man kan skippa tupelparenteserna om ett tupelargument är ensamt i sin parameterlista.

Lösn. uppg. 9. *Händelser.*

a) Den oföränderliga heltalsvariabeln `KeyPressed` i `introprog.PixelWindow.Event` har värdet 1.

b) Kodraden nedan tar hand om knappnedtryckningsfallet:

```
case PixelWindow.Event.KeyPressed => println(s"lastKey == \$w.lastKey")
```

c) När pil-upp-knappen på tangentbordet trycks ned får `w.lastKey` strängvärdet `"Up"`. Följande skrivs ut av testprogrammet när pil-upp-tangenten trycks ned och släpps upp:

```
1 lastEventType: 1 => KeyPressed
2 lastKey == Up
3 lastEventType: 2 => KeyReleased
4 lastKey == Up
```

d) En loop som låter användaren rita linjer med musen:

```
var start = (0,0)
while w.lastEventType != PixelWindow.Event.WindowClosed do
  w.awaitEvent(10)  // wait for next event for max 10 milliseconds
  w.lastEventType match {
    case PixelWindow.Event.MousePressed  =>
      start = w.lastMousePos

    case PixelWindow.Event.MouseReleased =>
      w.line(start._1, start._2, w.lastMousePos._1, w.lastMousePos._2)

    case PixelWindow.Event.WindowClosed  =>
       println("Goodbye!");
    case _ =>
  }
  PixelWindow.delay(100) // wait for 0.1 seconds
```

4.2 Extrauppgifter; träna mer

Lösn. uppg. 10. *Funktioner är objekt med en `apply`-metod.*Ja det går bra att skriva:

```
scala> plus(42, 43)
```

Kompilatorn fyller i `.apply` åt dig.

Lösn. uppg. 11. *Skapa moduler med hjälp av singelobjekt.*

a)

```
scala> "päronisglass".split('i')
val res0: Array[String] = Array(päron, sglass)
```

b)

```
scala> Test()
--- FREKVENSANALYS AV:
Fem     myror är fler än fyra elefanter. Ät gurka.
# bokstäver: 36
# ord      : 9
# meningar : 2

--- FREKVENSANALYS AV:
Galaxer i mina braxer. Tomat är gott. Päronsplitt.
# bokstäver: 40
# ord      : 8
# meningar : 3

--- FREKVENSANALYS AV:
Fem     myror är fler än fyra elefanter. Ät gurka. Galaxer i mina braxer. Tomat
är gott. Päronsplitt.
# bokstäver: 76
# ord      : 17
# meningar : 5
```

c) Objektet `statistics` har ett förändringsbart tillstånd i variabeln `history`. Tillståndet ändras vid anrop av `printFreq`.

d)

```
  object count:
    extension (s: String)
      def nbrOfLetters:Int = s.count(_.isLetter)
      def nbrOfWords:Int = split.words(s).size
      def nbrOfSentences: Int = split.sentences(s).size
```

Lösn. uppg. 12. *Tupler som parametrar.*

```
def distxy(x1: Int, y1: Int, x2: Int, y2: Int): Double =
  hypot(x1 - x2, y1 - y2)

def distpt(p1: (Int, Int), p2: (Int, Int)): Double =
  hypot(p1._1 - p2._1, p1._2 - p2._2)
```

```
def distp(p1: (Int, Int))(p2: (Int, Int)): Double =
  hypot(p1._1 - p2._1, p1._2 - p2._2)
```

Lösn. uppg. 13. *Tupler som funktionsresultat.*

```
def statistics(xs: Vector[Double]): (Int, Double, (Double, Double)) =
  (xs.size, xs.sum / xs.size, (xs.min, xs.max))
```

```
scala> statistics(Vector(0, 2.5, 5))
val res10: (Int, Double, (Double, Double)) = (3,2.5,(0.0,5.0))
```

Lösn. uppg. 14. *Skapa moduler med hjälp av paket.*

a)

```
> code paket.scala
> scala-cli paket.scala
> find . -type d         # linuxkommando som listar alla subkataloger
./.scala-build/project_103be31561-3d0d386400/classes
./.scala-build/project_103be31561-3d0d386400/classes/main
./.scala-build/project_103be31561-3d0d386400/classes/main/gurka
./.scala-build/project_103be31561-3d0d386400/classes/main/gurka/tomat
./.scala-build/project_103be31561-3d0d386400/classes/main/gurka/tomat/banan
./.scala-build/project_103be31561-3d0d386400/classes/main/gurka/tomat/banan/p2
./.scala-build/project_103be31561-3d0d386400/classes/main/gurka/tomat/banan/p2/
./.scala-build/project_103be31561-3d0d386400/classes/main/gurka/tomat/banan/p1
./.scala-build/project_103be31561-3d0d386400/classes/main/gurka/tomat/banan/p1/
./.scala-build/project_103be31561-3d0d386400/classes/main/gurka/tomat/banan/p1/
```

b)

```
> scala-cli run paket.scala --main-class gurka.tomat.banan.Main
Hej paket p1.p11!
Hej paket p1.p12!
Hej paket p2.p21!
```

c) Ja, i Scala 3 får paket ha variabler och funktioner på toppnivå.
https://stackoverflow.com/a/56566166

4.3 Fördjupningsuppgifter; utmaningar

Lösn. uppg. 15. *Hur klara sig utan* `do while` *i Scala 3?*

a) Det blir kompileringsfel:

```
> scala-cli repl --scala 3
Welcome to Scala 3.1.3 (17.0.3, Java OpenJDK 64-Bit Server VM).
Type in expressions for evaluation. Or try :help.

scala> var i = 0
var i: Int = 0

scala> do i += 1 while (i < 10)
-- [E103] Syntax Error: ----------------------------------------
```

```
1 |do i += 1 while (i < 10)
  |^^
  |Illegal start of statement
```

b)

```
> scala-cli repl --scala 3
Welcome to Scala 3.1.3 (17.0.3, Java OpenJDK 64-Bit Server VM).
Type in expressions for evaluation. Or try :help.

scala> var i = 0
var i: Int = 0

scala> while
     |   i += 1
     |   i < 10
     | do ()

scala> i
val res0: Int = 10
```

Lösn. uppg. 16. *Postfixa operatorer för inkrementering och dekrementering.*

```
extension (i: Int)
  def ++ = i + 1
  def -- = i - 1
```

Lösn. uppg. 17. *Använda färdigt paket: Färgväljare.*

a) Den valda färgen returneras efter att användaren tryckt **OK**

```
1 scala> introprog.Dialog.selectColor()
2 val res1: java.awt.Color = java.awt.Color[r=0,g=204,b=0]
```

b) Default-färgen röd returneras efter att användaren tryckt **Cancel**

c) Färgväljaren återgår till default-färgen.

Lösn. uppg. 18. *Använda färdigt paket: användardialoger.*

a)

```
1 scala> introprog.Dialog.show("Game over!")
```

b) Funktionen `input` returnerar en sträng som blir tomma strängen `""` om användaren klickar **Cancel**

```
1 scala> val name = introprog.Dialog.input("Vad heter du?")
2 name: String = Oddput Superkodare
```

c) Funktionen `select` returnerar en sträng med texten på knappen som användaren tryckte på.

```
1 scala> introprog.Dialog.select("Vad väljer du?",Vector("Sten","Sax","Påse"))
2 val res4: String = Påse
```

Lösn. uppg. 19. *Skapa din egen `jar`-fil.*

a)

`jar -create -verbose -file <namn på skapad jar-fil> <namn på det som ska packas>`

b)

```
package hello

object Main:
  def main(args: Array[String]): Unit = println("Hello package!")
```

```
scala-cli compile hello.scala --destination .
```

c)

```
> jar -c -v -f my.jar hello
> ls
> scala-cli repl --jar my.jar
scala> hello.Main.main(Array())
Hello package!
```

d)

```
> scala-cli run --jar my.jar  --main-class hello.Main
```

Lösn. uppg. 20. *Hur stor är JDK8?* Med JDK8-plattformen kommer 4240 färdiga klasser, som är organiserade i 217 olika paket. Se Stackoverflow:
http://stackoverflow.com/questions/3112882

5. Lösning `classes`

5.1 Grunduppgifter; förberedelse inför laboration

Lösn. uppg. 1. *Para ihop begrepp med beskrivning.*

klass	1	⇝	H	en mall för att skapa flera instanser av samma typ
instans	2	⇝	I	upplaga av ett objekt med eget tillståndsminne
konstruktor	3	⇝	M	skapar instans, allokerar plats för tillståndsminne
klassparameter	4	⇝	K	binds till argument som ges vid konstruktion
referenslikhet	5	⇝	B	instanser anses olika även om tillstånden är lika
innehållslikhet	6	⇝	J	instanser anses lika om de har samma tillstånd
case-klass	7	⇝	F	slipper skriva new; automatisk innehållslikhet
getter	8	⇝	L	indirekt åtkomst av attributvärde
setter	9	⇝	A	indirekt tilldelning av attributvärde
kompanjonsobjekt	10	⇝	D	ser privata medlemmar i klass med samma namn
fabriksmetod	11	⇝	E	hjälpfunktion för indirekt konstruktion
`null`	12	⇝	G	ett värde som ej refererar till någon instans
`new`	13	⇝	C	nyckelord vid direkt instansiering av klass

Lösn. uppg. 2. *Klass och instans.*

a)

`Singelpunkt.x`	1	⇝	B	`1`
`Punkt.x`	2	⇝	G	`value is not a member of object`
`val p  = new Singelpunkt`	3	⇝	C	`Not found: type`
`val p1 = new Punkt`	4	⇝	D	`p1: Punkt = Punkt@27a1a53c`
`val p2 = Punkt()`	5	⇝	F	`p2: Punkt = Punkt@51ab04bd`
`{ p1.x = 1; p2.x }`	6	⇝	E	`3`
`(new Punkt).y`	7	⇝	H	`2`
`{ val p: Punkt = null; p.x }`	8	⇝	A	`java.lang.NullPointerException`

b)

fel	*typ*	*förklaring*
`value is not a member of object`	kompileringsfel	det finns ingen instans med namnet `Punkt`. *Felmeddelandet syftar på att det i klassens autogenererade konstruktor-ombud saknas en variabel med namnet* `x`.
`Not found: type`	kompileringsfel	det finns ingen klass som heter `Singelpunkt`
`NullPointerException`	körtidsfel	det går inte att referera attribut i en instans som inte finns

Lösn. uppg. 3. *Klassparametrar.*

a)

`val p1 = Point(1, 2)`	1	⇝ C	`p1: Point = Point@30ef773e`
`val p2 = Point()`	2	⇝ A	`missing argument for parameter`
`val p2 = Point(3, 4)`	3	⇝ E	`p2: Point = Point@218cf600`
`p2.x - p1.x`	4	⇝ B	`2`
`Point(0, 1).y`	5	⇝ F	`1`
`Point(0, 1, 2)`	6	⇝ D	`too many arguments for constructor`

b)

fel	*typ*	*förklaring*
`missing argument for parameter`	kompileringsfel	du måste ge argument vid konstruktion av klassen `Point`
`too many arguments for constructor`	kompileringsfel	antalet argument stämmer ej överens med antalet klassparametrar

Lösn. uppg. 4. *Oföränderlig klass med defaultargument.*

a)

`val p1 = Point3D()`	1	⇝ C	`p1: Point3D = Point3D@2eb37eee`
`val p2 = Point3D(y = 1)`	2	⇝ F	`p2: Point3D = Point3D@65a9e8d7`
`Point3D(z = 2).z`	3	⇝ E	`value cannot be accessed`
`p2.y = 0`	4	⇝ B	`Reassignment to val`
`p2.y == 0`	5	⇝ A	`false`
`p1.x == Point3D().x`	6	⇝ D	`true`

b) Problemet är att så som klassen `Point3D` är deklarerad går det inte att avläsa z-koordinaten efter att en instans konstruerats. Det vore bättre om även z-attributet är `val`.

Lösn. uppg. 5. *Case-klass, `this`, likhet, `toString` och kompanjonsobjekt.*

a)

```
val p1 = Pt(1, 2)
val p2 = Pt(y = 3)
val p3 = MutablePt(5, 6)
val p4 = Mutable()
p2.moved(dx = 1) == Pt(1, 3)
p3.move(dy = 1) == MutablePt(5, 7)
```

1	⇝	E	`Pt(1,2)`
2	⇝	C	`Pt(0,3)`
3	⇝	A	`MPt(5,6)`
4	⇝	D	`Not found`
5	⇝	F	**`true`**
6	⇝	B	**`false`**

b) Kompilatorn härleder `MutablePt` eftersom det är typen på självreferensen this.

```
scala> :type new MutablePt().move()
MutablePt
```

c) Instansiering med universella apply-metoder (eng. *universal apply methods*) är godis som gör koden enklare att läsa och skriva. Detta är möjligt tack vare att det vid kompilering automatiskt skapas ett konstruktor-ombud (eng. *constructor proxy*) som instansierar objektet med nyckelordet **`new`**. Ett konstruktor-ombud är ett kompanjonsobjekt med tillhörande apply-metod.

Ett fall då **`new`** uttryckligen måste användas är vid implementering av egen apply-metod i ett kompanjonsobjekt. Om **`new`** inte används inuti apply-metoden, kommer samma metod att anropas rekursivt istället för att en ny instans skapas. Se följande exempel:

```
class Point3D(val x: Int, val y: Int, val z: Int)

object Point3D:
  var secretNumber = 42
  def apply(x: Int, y: Int, z: Int): Point3D =
    if secretNumber == 42 then
      Point3D(x, y, z) // Koden kommer fastna i en evig loop.

    else new Point3D(x, y, z) // Funkar eftersom 'new' används.
```

d) En metod som avläser (delar av) ett objekts (privata) tillstånd utan att ändra det kallas för en *getter*.

Lösn. uppg. 6. *Implementera delar av klasserna `Pos`, `KeyControl`, `Mole` och `BlockWindow` som behövs under laborationen `blockbattle1`.*Denna uppgift är laborationsförberedelse. Utvärdera dina lösningar genom egna tester i REPL.

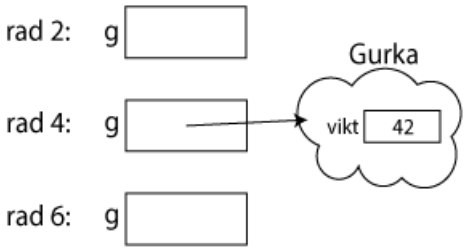
a) Det går inte att anropa `Pos.moved(0,1)`. Anledningen till detta är att `moved` inte existerar i kompanjonsobjektet `Pos`, därav felmeddelandet ”value `moved` is not a member of object `Pos`”. För att anropa en metod definierad inuti en klass måste man göra anropet via en (referens till en) instans av klassen.

5.2 Extrauppgifter; träna mer

Lösn. uppg. 7. *Instansiering med **`new`** och värdet **`null`**.*

a) Rad 3 och 7 ger båda felmeddelandet `java.lang.NullPointerException`, på grund av försök att referera medlemmar med hjälp av en **`null`**-referens, som alltså inte pekar på något objekt.

b)



Lösn. uppg. 8. *Skapa en punktklass som kan hantera polära koordinater.*

a)

```
package graphics

case class Point(x: Double, y: Double):
  val r: Double          = math.hypot(x, y)
  val theta: Double      = math.atan2(y, x)
  def negY: Point        = Point(x, -y)
  def +(p: Point): Point = Point(x + p.x, y + p.y)

object Point:
  def polar(r: Double, theta: Double): Point =
    Point(r * math.cos(theta), r * math.sin(theta))
```

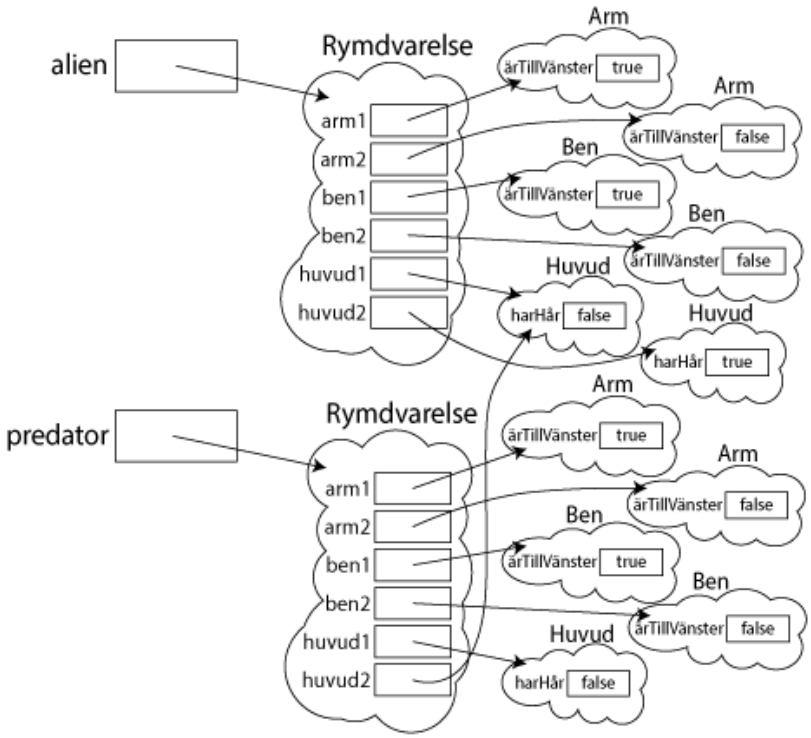
b) TODO!!!

c) TODO!!!

d) TODO!!!

Lösn. uppg. 9. *Klasser, instanser och skräp.*

a) Vi skapar två rymdvarelser, `alien` och `predator`, med vardera två ben och två armar, samt vardera två huvuden (där det ena är skalligt och det andra har hår). Efter det är varken `alien` eller `predator` skallig eftersom båda har ett huvud med hår. Sen låter man referensen till `predators` huvud med hår referera till aliens huvud utan hår. Nu är predator helt skallig och delar huvud med alien.



b) Eftersom det inte längre finns någon referens som pekar på det objektet kommer skräpsamlaren att ta hand om det och det kommer förr eller senare skrivas över av något annat när platsen i minnet behövs. Objekt som inte har någon referens till sig går inte att komma åt.

Lösn. uppg. 10. *Case-klass. Oföränderlig kvadrat.*

a)

```
case class Square(val x: Int = 0, val y: Int = 0, val side: Int = 1):
  val area: Int = side * side

  def moved(dx: Int, dy: Int): Square = Square(x + dx, y + dy, side)

  def isEqualSizeAs(that: Square): Boolean = this.side == that.side

  def scale(factor: Double): Square =
    Square(x, y, (side * factor).round.toInt)

object Square:
  val unit: Square = Square()
```

b)

```
scala> val (s1, s2) = (Square(), Square(1, 10, 1))
val s1: Square = Square(0,0,1)
val s2: Square = Square(1,10,1)

scala> val s3 = s1 moved (1,-5)
```

```
6  val s3: Square = Square(1,-5,1)
7
8  scala> s1 isEqualSizeAs s3       // lika storlek
9  val res0: Boolean = true
10
11 scala> s2 isEqualSizeAs s1       // lika storlek
12 val res1: Boolean = true
13
14 scala> s1 isEqualSizeAs Square.unit   // s1 har sidan 1
15 val res2: Boolean = true
16
17 scala> s2.scale(math.Pi) isEqualSizeAs s2  // olika storlek
18 val res3: Boolean = false
19
20 scala> s2.scale(math.Pi) == s2.scale(math.Pi) // lika innehåll
21 val res4: Boolean = true
22
23 scala> s2.scale(math.Pi) eq s2.scale(math.Pi)  // olika objekt
24 val res5: Boolean = false
25
26 scala> Square.unit eq Square.unit   // samma objekt
27 val res6: Boolean = true
```

5.3 Fördjupningsuppgifter; utmaningar

Lösn. uppg. 11. *Innehållslikhet mellan olika typer.*

```
scala> 42 == "Fyrtiotvå"
1 |42 == "Fyrtiotvå"
  |^^^^^^^^^^^^^^^^^
  |Values of types Int and String cannot be compared with == or !=

scala> Gurka(50) == Bil("Sedan")
val res0: Boolean = false
```

Det andra uttrycket är problematiskt eftersom det alltid kommer resultera i `false`, då klasserna `Gurka` och `Bil` är två ojämförbara typer som inte bör jämföras med avseende på innehållslikhet. Detta försämrar typsäkerheten vilket ökar risken för svårupptäckta buggar där fel typer jämförs.

Likhetsjämförelser som sker mellan primitiva typer typkollas av kompilatorn och kan därför ge kompileringsfel om två olika typer, såsom `Int` och `String`, jämförs med varandra. Detta gäller dock i regel inte egendefinierade typer, vilket alltså innebär att en likhetsjämförelse mellan olika egendefinierade typer alltid resulterar i `false`.

Det är emellertid möjligt att få samma typkoll för egendefinierade typer som för primitiva typer genom att importera `scala.language.strictEquality`.

```
import scala.language.strictEquality
class Gurka(val vikt: Int)

class Bil(val typ: String)
```

```
scala> Gurka(50) == Bil("Sedan")
1 |Gurka(50) == Bil("Sedan")
  |^^^^^^^^^^^^^^^^^^^^^^^^
  |Values of types Gurka and Bil cannot be compared with == or !=
```

Lösn. uppg. 12. *Attributrepresentation. Privat konstruktor. Fabriksmetod.*

a) Det blir kompileringsfel eftersom konstruktorn är privat.

```
scala> class Point private (val x: Int, val y: Int)
     | object Point:
     |   def apply(x: Int = 0, y: Int = 0): Point = new Point(x, y)
     |   val origo = apply()
     |
// defined class Point
// defined object Point

scala> new Point(0, 0)
1 |new Point(0, 0)
  |    ^^^^^
  |constructor Point cannot be accessed as a member of Point from module class
```

b)

- Genom att ha en privat konstruktor och bara göra indirekt instansiering via fabriksmetod är lätt ändra attributrepresentation i framtiden utan att befintlig kod behöver ändras.

- Accessreglerna för kompanjonsobjekt är sådana att kompanjoner ser varandras privata delar.

c)

```
class Point private (private val p: (Int, Int)):
  def x: Int = p._1
  def y: Int = p._2

object Point:
  def apply(x: Int = 0, y: Int = 0): Point = new Point(x, y)
  val origo = apply()
```

Lösn. uppg. 13. *Synlighet av klassparametrar och konstruktor,* ***`private`****`[this]`*.

a) `Gurka5` är trasig. Eftersom vikten i `Gurka5` är privat för instansen och inte klassen, kan en instans inte accessa en annan instans vikt.

```
11 |  def kompisVikt = kompis.vikt
   |                   ^^^^^^^^^^^
   |value vikt cannot be accessed as a member of (Gurka5.this.kompis : Gurka5)
```

b)

```
scala> new Gurka1(42).vikt
1 |new Gurka1(42).vikt
  |^^^^^^^^^^^^^^^^^^^
  |value vikt cannot be accessed as a member of Gurka1 from module class

scala> new Gurka2(42).vikt
val res0: Int = 42

scala> new Gurka3(42).vikt
1 |new Gurka3(42).vikt
  |^^^^^^^^^^^^^^^^^^^
  |value vikt cannot be accessed as a member of Gurka3 from module class

scala> val ingenGurka: Gurka4 = null
val ingenGurka: Gurka4 = null

scala> new Gurka4(42, ingenGurka).kompisVikt
java.lang.NullPointerException: Cannot invoke "rs$line$1$Gurka4.vikt()" bec...
  at rs$line$1$Gurka4.kompisVikt(rs$line$1:8)
  ... 38 elided

scala> new Gurka4(42, new Gurka4(84, null)).kompisVikt
val res2: Int = 84

scala> new Gurka6(42)
1 |new Gurka6(42)
  |    ^^^^^^
  |constructor Gurka6 cannot be accessed as a member of Gurka6 from module...

scala> new Gurka7(-42)
1 |new Gurka7(-42)
  |    ^^^^^^
```

```
  |constructor Gurka7 cannot be accessed as a member of Gurka7 from module...

scala> Gurka7(-42)
java.lang.IllegalArgumentException: requirement failed: negativ vikt: -42

scala> val g = Gurka7(42)
val g: Gurka7 = Gurka7@51fd1c7c

scala> g.vikt
val res4: Int = 42

scala> g.vikt = -1

scala> g.vikt
val res5: Int = -1
```

Lösn. uppg. 14. *Egendefinierad setter kombinerat med privat konstruktor.*

a)

Rad 1:

```
java.lang.IllegalArgumentException: requirement failed: negativ vikt: -42
```

`Gurka8.apply` kräver att `vikt >= 0` annars kastar `require` ett undantag.

Rad 5:

```
java.lang.IllegalArgumentException: requirement failed: negativ vikt: -1
```

Settern `vikt_=` kräver att `vikt >= 0` annars kastar `require` ett undantag.

Rad 7:

```
java.lang.IllegalArgumentException: requirement failed: negativ vikt: -958
```

Eftersom `42 - 1000` är mindre än noll kastar `require` ett undantag.

b) Man kan sätta egna mer specifika krav på vad som får göras med värdena så man har större koll på att inget oväntat händer.

Lösn. uppg. 15. *Objekt med föränderligt tillstånd (eng. mutable state).*

a)

```
class Frog private (initX: Int = 0, initY: Int = 0):
  private var _x: Int = initX
  private var _y: Int = initY
  private var _distanceJumped: Double = 0

  def x: Int = _x
  def y: Int = _y

  def jump(dx: Int, dy: Int): Unit =
    _x += dx
    _y += dy
    _distanceJumped += math.hypot(dx, dy)
```

```
  def randomJump: Unit =
    def rnd = util.Random.nextInt(10) + 1
    jump(rnd, rnd)

  def distanceToStart: Double = math.hypot(x,y)
  def distanceJumped: Double = _distanceJumped
  def distanceTo(f: Frog): Double = math.hypot(x - f.x, y - f.y)

object Frog:
  def spawn(): Frog = Frog()
```

b) Exempel på testprogram:

```
object FrogTest:
  def test(): Unit =
    val f1 = Frog.spawn()
    assert(f1.x == 0 && f1.y == 0, "Test of spawn, reqt 1 & 4 failed.")

    f1.jump(4, 3)
    assert(f1.x == 4 && f1.y == 3, "Test of jump, reqt 1 & 4 failed.")

    f1.jump(4, 3)
    assert(f1.distanceJumped == 10, "Test of jump, reqt 2 failed.")

    f1.jump(-4, -3)
    assert(f1.distanceToStart == 5, "Test of jump, reqt 3 failed.")

    for x <- 1 to 10000 do
      val f2 = Frog.spawn()
      f2.randomJump
      assert(f2.x > 0 && f2.x <= 10 && f2.y > 0 && f2.y <= 10,
        "Test of randomJump, reqt 5 failed.")

    println("Test Ok!")
```

c) En metod som är en indirekt avläsning av attributvärden kallas getter.

d)

```
class Frog private (initX: Int = 0, initY: Int = 0):
  private var _x: Int = initX
  private var _y: Int = initY
  private var _distanceJumped: Double = 0

  def jump(dx: Int, dy: Int): Unit =
    _x += dx
    _y += dy
    _distanceJumped += math.hypot(dx, dy)

  def x: Int = _x
  def x_=(newX: Int): Unit = // Setter för x
    _distanceJumped += math.abs(x - newX)
```

```
    _x = newX

  def y: Int = _y
  def y_=(newY: Int): Unit = // Setter för y
    _distanceJumped += math.abs(y - newY)
    _y = newY


  def randomJump: Unit =
    def rnd = util.Random.nextInt(10) + 1
    jump(rnd, rnd)

  def distanceToStart: Double = math.hypot(x,y)
  def distanceJumped: Double = _distanceJumped
  def distanceTo(f: Frog): Double = math.hypot(x - f.x, y - f.y)

object Frog:
  def spawn(): Frog = Frog()
```

e)

```
object FrogSimulation:
  def isAnyCollision(frogs: Vector[Frog]): Boolean =
    var found = false
    frogs.indices.foreach(i =>  // generate all pairs (i,j)
      for j <- i + 1 until frogs.size do
        if !found then
          found = frogs(i).distanceTo(frogs(j)) <= 0.5
    )
    found

  def jumpUntilCrash(n: Int = 100, initDist: Int = 8): (Int, Double) =
    val frogs = Vector.fill(n)(Frog.spawn())
    (0 until n).foreach(i => frogs(i).x = i * initDist)
    var count = 0
    while !isAnyCollision(frogs) do
      frogs(util.Random.nextInt(n)).randomJump
      count += 1
    (count, frogs.map(_.distanceJumped).sum)


  def run(nbrOfCrashTests: Int = 10) =
    for i <- 1 to nbrOfCrashTests do
      val (n, dist) = jumpUntilCrash()
      println(s"\nAntalet looprundor innan grodkrock: $n")
      println(s"Totalt avstånd hoppat av alla grodor: $dist")
```

Lösn. uppg. 16. *Objekt med föränderligt tillstånd (eng. mutable state).*

```
class Square private (val initX: Int, val initY: Int, val initSide: Int):
  private var nMoves = 0
```

```
  private var sumCost = 0.0

  private var _x = initX
  private var _y = initY

  private var _side = initSide

  private def addCost(): Unit =
    sumCost += math.hypot(x - initX, y - initY) * side

  def x: Int = _x
  def y: Int = _y

  def side = _side

  def scale(factor: Double): Unit = _side = (_side * factor).round.toInt

  def move(dx: Int, dy: Int): Unit =
    _x += dx; _y += dy
    nMoves += 1
    addCost()

  def moveTo(x: Int, y: Int): Unit =
    _x = x; _y = y
    nMoves += 1
    addCost()

  def cost: Double = sumCost

  def pay: Double = {val temp = sumCost; sumCost = 0; temp}

  override def toString: String =
    s"Square[($x, $y), side: $side, #moves: $nMoves times, cost: $sumCost]"

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

  def apply(x: Int, y: Int, side: Int): Square =
    require(side >= 0, s"side must be positive: $side")
    val sq = (new Square(x, y, side))
    created :+= sq
    sq

  def apply(): Square = apply(0, 0, 1)

  def totalNumberOfMoves: Int = created.map(_.nMoves).sum

  def totalCost: Double = created.map(_.cost).sum
```

6. Lösning `patterns`

6.1 Grunduppgifter; förberedelse inför laboration

Lösn. uppg. 1. *Matcha på konstanta värden.*

a) Scalas `match`-uttryck jämför stegvis värdet med varje `case` för att sedan returnera ett värde tillhörande motsvarande `case`.

b)

```
scala.MatchError
```

Exekveringsfel, uppstår av en viss input under körningen.

Lösn. uppg. 2. *Gard i case-grenar.*

Garden som införts vid `case 'g'` slumpar fram ett tal mellan 0 och 1 och om talet inte är större än 0.5 så blir det ingen matchning med `case 'g'` och programmet testar vidare tills default-caset.
Gardens krav måste uppfyllas för att det ska matcha som vanligt.

Lösn. uppg. 3. *Mönstermatcha på attributen i case-klasser.*
G100true. Vid byte av plats: Gtrue100.
`match` testar om kompanjonsobjektet `Gurka` är av typen `Gurka` med två parametervärden. De angivna parametrarna tilldelas namn, `vikt` får namnet `v` och `ärRutten` namnet `rutten` och skrivs sedan ut. Byts namnen dessa ges skrivs de ut i den omvända ordningen.

Lösn. uppg. 4. *Matcha på case-objekt och nyttan med `sealed`.*

a)

```
Cannot extend sealed trait Färg in a different source file
```

Felmeddelandet fås av att REPL:en behandlar varje inmatning individuellt och tillåter därför inte att subtypen `Spader` ärver från (eng. *extends*) supertypen `Färg` eftersom denna var förseglad (eng. *sealed*). Mer om detta senare i kursen...

b) -

c) Förusatt att `import Kortlek._` har skrivits...

```
def parafärg(f: Färg): Färg = f match
  case Spader  => Klöver
  case Hjärter => Ruter
  case Ruter   => Hjärter
  case Klöver  => Spader
```

d)

```
<console>:17: warning: match may not be exhaustive.
It would fail on the following input: Ruter
```

Varningen kommer redan vid kompilering.

e)

```
scala.MatchError: Ruter (of class Ruter)
  at .parafärg(<console>:17)
```

Detta är ett körtidsfel.

f) Om en klass är `sealed` innebär det att om ett element ska matchas och är en subtyp av denna klass så ger Scala varning redan vid kompilering om det finns en risk för ett `MatchError`, alltså om `match`-uttrycket inte är uttömmande och det finns fall som inte täcks av ett `case`.
En förseglad supertyp innebär att programmeraren redan vid kompileringstid får en varning om ett fall inte täcks och i sånt fall vilket av undertyperna, liksom annan hjälp av kompilatorn. Detta kräver dock att alla subtyperna delar samma fil som den förseglade klassen.

Lösn. uppg. 5. *Mönstermatcha enumeration.*a)

```
def parafärg(f: Färg): Färg = f match
  case Färg.Spader  => Färg.Klöver
  case Färg.Hjärter => Färg.Ruter
  case Färg.Ruter   => Färg.Hjärter
  case Färg.Klöver  => Färg.Spader
```

Likt uppgift **????** så kan även här en `import`-sats skrivas för att nå medlemmarna i `Färg` utan punktnotation. Det är dock inte alltid fördelaktigt att importera medlemmar till den globala namnrymden, då det kan förekomma namnkrockar. Anta ett exempel där vi jobbar på ett program med grafiskt användargränssnitt där vi har en färg `Red` definerad. Anta också att vi nu till vårt program vill importera ytterligare en röd färg för kulörerna hjärter och ruter, denna också namngiven `Red`. I detta scenario hade det uppstått en namnkrock då `Red` redan är definerad så importeringen hade ej kunnat ske.

b) Vid mönstermatchning så fungerar `sealed trait` ihop med `case`-objekt i praktiken likadant som att använda sig av `enum`. Vi såg att i deluppgift **????** så varnade REPL redan vid kompilering att denna matchning inte var uttömmande (eng. *exhaustive*). Detta gäller även vid användning av `enum`.

Lösn. uppg. 6. *Betydelsen av små och stora begynnelsebokstäver vid matchning.*

a) Både `str` och `vadsomhelst` matchar med inputen, oavsett vad denna är på grund av att de har en liten begynnelsebokstav.
`str` har dock en gard att strängen måste börja med *g* vilket gör så endast `val g = "gurka"` matchar med denna. `val x = "urka"` plockas dock upp av `vadsomhelst` som är utan gard.

b)

```
1 <console>:16: warning: patterns after a variable pattern cannot match (SLS 8.1
2 .1)
```

och

```
1 <console>:17: warning: unreachable code due to variable patter 'tomat' on line
2 16
```

Trots att en klass `tomat` existerar så tolkar Scalas `match` den som en `case`-gren som fångar allt på grund av en liten begynnelsebokstav. Detta gör så alla objekt som inte är av typen `Gurka` kommer ge utskriften *tomat* och att sista caset inte kan nås.

c)

```
case `tomat` => println("tomat")
```

Lösn. uppg. 7. *Matcha på innehåll i en Vector.*

```
jeh
jed
42
```

För varje element i `xss` görs en matching som resulterar i en sträng. Vad som händer i varje gren förklaras nedan.

1. Första match-grenen aktiveras aldrig eftersom `xss` ej innehåller någon tom vektor.
2. Andra grenen passar med `Vector("hej")` och variablen `a` binds till `"hej"`.
3. Tredje grenen matchar `Vector("på", "dej")` där första värdet binds inte till någon variabel eftersom understreck finns på motsvarande plats, medan andra värdet binds till `b`.
4. Fjärde grenen matchar en sekvens med tre värden där mittenvärdet är `"x"`. Den sista grenen aktiveras inte i detta exempel men hade matchat allt som inte fångas av tidigare grenar.

Lösn. uppg. 8. *Använda `Option` och matcha på värden som kanske saknas.*

a)

1. `var kanske` blir en `Option` som håller `Int` men är utan något värde, kallas då `None`.
2. Eftersom `var kanske` är utan värde är storleken av den 0.
3. `var kanske` tilldelas värdet 42 som förvaras i en `Some` som visar att värde finns.
4. Eftersom `var kanske` nu innehåller ett värde är storleken 1.
5. Eftersom `var kanske` innehåller ett värde är den inte tom.
6. Eftersom `var kanske` innehåller ett värde är den definierad.
7. `def ökaOmFinns` matchar en `Option[Int]` med dess olika fall.
   Finns ett värde, alltså `opt: Option[Int]` är en `Some`, så returneras en `Some` med ursprungliga värdet plus 1.
   Finns inget värde, alltså `opt: Option[Int]` är en `None`, så returneras en `None`.
8. -
9. -
10. -
11. `def ökaOmFinns` appliceras på `kanske` och returnerar en `Some` med värdet hos `kanske` plus 1, alltså 43.
12. `def öka` tar emot värdet av en `Int` och returnerar värdet av denna plus 1.

13. `map` applicerar `def öka` till det enda elementen i `kanske`, 42. Denna funktion returnerar en `Some` med värdet 43 som tilldelas `merKanske`.

b)

1. `val meningen` blir en `Some` med värdet 42.
2. `val ejMeningen` blir en `Option[Int]` utan något värde, en `None`.
3. `map(_ + 1)` appliceras på `meningen` och ökar det existerande värdet med 1 till 43.
4. `map(_ + 1)` appliceras på `ejMening` men eftersom inget värde existerar fortsätter denna vara `None`.
5. `map(_ + 1)` appliceras ännu en gång på `ejMening` men denna gång inkluderas metoden `orElse`. Om ett värde inte existerar hos en `Option`, alltså är av typen `None`, så utförs koden i `orElse`-metoden som i detta fall skriver ut *saknas* för värdet som saknas.
6. Samma anrop från föregående rad utförs denna gång på `meningen` och eftersom ett värde finns utförs endast första biten som ökar detta värde med 1.

Denna metod kan användas i stället för `match`-versionen i föregående exempel i och med dennas simplare form. En `Option` innehåller ju antingen ett värde eller inte så ett längre `match`-uttryck är inte nödvändigt.

c)

1. En vektor `xs` skapas med var femte tal från 42 till 82.
2. En tom `Int`-vektor `e` skapas.
3. `headOption` tar ut första värdet av vektorn `xs` och returnerar den sparad i en `Option`, `Some(42)`.
4. Första värdet i vektorn `xs` sparas i en `Option` och hämtas sedan av `get`-metoden, 42.
5. Som i föregående rad men denna gång används `getOrElse` som om den `Option` som returneras saknar ett värde, alltså är av typen `None`, returnerar 0 istället. Eftersom `xs` har minst ett värde så är den `Option` som returneras inte `None` och ger samma värde som i föregående, 42.
6. Som föregående rad fast istället för att returnera 0 om värde saknas så returneras en `Option[Int]` med 0 som värde.
7. `headOption` försöker ta ut första värdet av vektorn `e` men eftersom denna saknar värden returneras en `None`.
8.
```
1 java.util.NoSuchElementException: None.get
```

   Liksom föregående rad returnerar `headOption` på den tomma vektorn `e` en `None`. När `get`-metoden försöker hämta ett värde från en `None` som saknar värde ger detta upphov till ett körtidsfel.

9. Liksom i föregående returneras `None` av `headOption` men eftersom `getOrElse`-metoden används på denna `None` returneras 0 istället.

10. Liksom föregående används `getOrElse`-metoden på den `None` som returneras. Denna gång returneras dock en `Option[Int]` som håller värdet 0.

11. En vektor innehållandes elementen `xs`-vektorn och 3 `e`-vektorer skapas.

12. `map` använder metoden `lastOption` på varje delvektor från vektorn på föregående rad. Detta sammanställer de sista elementen från varje delvektor i en ny vektor. Eftersom vektor `e` är tom returneras `None` som element från denna.

13. Samma sker som i föregående rad men `flatten`-metoden appliceras på slutgiltiga vektorn som rensar vektorn på `None` och lämnar endast faktiska värden.

14. `lift`-metoden hämtar det eventuella värdet på plats 0 i `xs` och returnerar den i en `Option` som blir `Some(42)`.

15. `lift`-metoden försöker hämta elementet på plats 1000 i `xs`, eftersom detta inte existerar returneras `None`.

16. Samma sker som i föregående fast applicerat på vektorn `e`. Sedan appliceras `getOrElse(0)` som, eftersom `lift`-metoden returnerar `None`, i sin tur returnerar 0.

17. `find`-metoden anropas på `xs`-vektorn. Den letar upp första talet över 50 och returnerar detta värde i en `Option[Int]`, alltså `Some(52)`.

18. `find`-metoden anropas på `xs`-vektorn. Den letar upp första värdet under 42 men eftersom inget värde existerar under 42 i `xs` returneras `None` istället.

19. `find`-metoden anropas på `e`-vektorn och skriver ut *HITTAT!* om ett element under 42 hittas. Eftersom `e`-vektorn är tom returneras `None` vilket `foreach` inte räknar som element och därav inte utförs på.

d) Användning av -1 som returvärde vid fel eller avsaknad på värde kan ge upphov till körtidsfel som är svåra att upptäcka. `null` kan i sin tur orsaka kraschar om det skulle bli fel under körningen. `Option` har inte samma problem som dessa, används ett `getOrElse`-uttryck eller dylikt så kraschar inte heller programmet.
Dessutom behöver inte en funktion som returnerar en `Option` samma dokumentation av returvärdena. Istället för att skriva kommentarer till koden på vilka värden som kan returneras och vad dessa betyder så syns det direkt i koden.
Slutgiltligen är `Option` mer typsäkert än `null`. När du returnerar en `Option` så specificeras typen av det värde som den kommer innehålla, om den innehåller något, vilket underlättar att förstå och begränsar vad den kan returnera.

Lösn. uppg. 9. *Kasta undantag.*

a)

1. Ett `Exception` kastas med felmeddelandet *PANG!*.

2. Flera olika typer av `Exception` visas.

3. En typ av `Exception`, `IllegalArgumentException`, kastas med felmeddelandet *fel fel fel*.

4. Ett undantag med felmeddelandet `stormvind!` kastas och fångas av **`catch`**-uttrycket. Ett **`match`**-uttryck undersöker undantaget och skriver ut meddelandet, samt returnerar -1.

b) Exempelvis:
`OutOfMemoryError`, om programmet får slut på minne.
`IndexOutOfBoundsException`, om en vektorposition som är större än vad som finns hos vektorn försöker nås.
`NullPointerException`, om en metod eller dylikt försöker användas hos ett objekt som inte finns och därav är en nullreferens.

c) om både try-grenen och catch-grenen har samma typ, här `Int`, så härleder kompilatorn samma typ för hela uttrycket. Skulle **`catch`**-grenen returnera ett värde av en helt annan typ istället, t.ex. `String`, så blir den mest precisa typen som kompilatorn kan härleda för hela uttrycket `Matchable`, som är en direkt subtyp till den mest generella typen `Any`.

Lösn. uppg. 10. *Fånga undantantag med `scala.util.Try`.*

a)

1. **`def`** `pang` skapas som kastar ett `Exception` med felmeddelandet *PANG!*.
2. Scalas verktyg `Try`, `Success` och `Failure` importeras.
3. **`def`** `pang` anropas i `Try` som fångar undantaget och kapslar in den i en `Failure`.
4. Metoden `recover` matchar undantaget i `Failure` från föregående rad med ett **`case`** och gör om föredetta `Failure` till `Success` vid matchning, liknande **`catch`**.
5. Strängen *tyst* körs i föregående test men eftersom inget undantag kastas blir den inkapslad i en `Success` och `recover` behöver inte göra något. Den tar endast hand om undantag.
6. **`def`** `kanskePang` skapas som har lika stor chans att returnera strängen *tyst* såsom anropa **`def`** `pang`.
7. **`def`** `kanskeOk` skapas som testar **`def`** `kanskePang` med `Try`.
8. En vektor `xs` fylls med resultaten, `Success` och `Failure`, från 100 körningar av `kanskeOk`.
9. Elementet på plats 13 i vektor `xs` matchas med något av 2 **`case`**. Om det är en `Success` skrivs *:)* ut, om en `Failure` skrivs *:(* plus felmeddelandet ut.
10. -
11. -
12. Metoden `isSuccess` testar om elementet på plats 13 i `xs` är en `Success` och returnerar **`true`** om så är fallet.
13. Metoden `isFailure` testar om elementet på plats 13 i `xs` är en `Failure` och returnerar **`true`** om så är fallet.
14. Metoden `count` räknar med hjälp av `isFailure` hur många av elementen i `xs` som är `Failure` och returnerar detta tal.

15. Metoden `find` letar upp med hjälp av `isFailure` ett element i `xs` som är `Failure` och returnerar denna i en `Option`.
16. `badOpt` tilldelas den första `Failure` som hittas i `xs`.
17. `goodOpt` tilldelas den första `Success` som hittas i `xs`.
18. Resultatet badOpt skrivs ut, `Option[scala.util.Try[String]] = Some(Failure(java.lang.Exception: PANG!))`
19. Metoden `get` hämtar från `badOpt` den `Failure` som förvaras i en `Option`.
20. Metoden `get` anropas ännu en gång på resultatet från föregående rad, alltså en `Failure`, som hämtar undantaget från denna och som då i sin tur kastas.
21. Metoden `getOrElse` anropas på den `Failure` som finns i `badOpt`. Eftersom detta är en `Exception` utförs `orElse`-biten istället för att undantaget försöker hämtas. Då returneras strängen *bomben desarmerad!*.
22. Metoden `getOrElse` anropas på den `Success` som finns i `goodOpt`. Eftersom detta är en `Success` med en normal sträng sparad i sig returneras denna sträng, *tyst*.
23. Metoden från föregående används denna gång på alla element i `xs` där resultatet skrivs ut för varje.
24. Metoden `toOption` appliceras på alla `Success` och `Failure` i `xs`. De med ett exception, alltså `Failure`, blir en `None` medan de med värden i `Success` ger en `Some` med strängen *tyst* i sig.
25. Metoden `flatten` appliceras på vektorn fylld med `Option` från föregående rad för att ta bort alla `None`-element.
26. Metoden `size` används på slutgiltiga listan från föregående rad för att räkna ut hur många `Some` som resultatet innehåller. Den har alltså beräknat antalet element i `xs` som var av typen `Success` med hjälp av `Option`-typen.

b) `pang` har returtypen `Nothing`, en specialtyp inom Scala som inte är kopplad till `Any`, och som inte går att returnera.

c) Typen `Nothing` är en subtyp av varenda typ i Scalas hierarki. Detta innebär att den även är en subtyp av `String` vilket implicerar att `String` inkluderar både strängar och `Nothing` och därav blir returtypen.

6.2 Fördjupningsuppgifter; utmaningar

Lösn. uppg. 11. *Använda matchning eller dynamisk bindning?*

a)

```
package vegopoly

trait Grönsak:
  def vikt: Int
  def ärRutten: Boolean
  def ärÄtbar: Boolean
```

```
case class Gurka(vikt: Int, ärRutten: Boolean) extends Grönsak:
  val ärÄtbar: Boolean = (!ärRutten && vikt > 100)

case class Tomat(vikt: Int, ärRutten: Boolean) extends Grönsak:
  val ärÄtbar: Boolean = (!ärRutten && vikt > 50)

object Main:
  def slumpvikt: Int = (math.random()*500 + 100).toInt

  def slumprutten: Boolean = math.random() > 0.8

  def slumpgurka: Gurka = Gurka(slumpvikt, slumprutten)

  def slumptomat: Tomat = Tomat(slumpvikt, slumprutten)

  def slumpgrönsak: Grönsak =
    if math.random() > 0.2 then slumpgurka else slumptomat

  def main(args: Array[String]): Unit =
    val skörd = Vector.fill(args(0).toInt)(slumpgrönsak)
    val ätvärda = skörd.filter(_.ärÄtbar)
    println("Antal skördade grönsaker: " + skörd.size)
    println("Antal ätvärda grönsaker: " + ätvärda.size)
```

b) Följande `case class` läggs till:

```
case class Broccoli(vikt: Int, ärRutten: Boolean) extends Grönsak:
  val ärÄtbar: Boolean = (!ärRutten && vikt > 80)
```

Därefter läggs följande till i `object Main` innan `def slumpgrönsak`:

```
def slumpbroccoli: Broccoli = Broccoli(slumpvikt, slumprutten)
```

Slutligen ändras `def slumpgrönsak` till följande:

```
def slumpgrönsak: Grönsak =     // välj t.ex. denna fördelning:
  val rnd = math.random()
  if rnd > 0.5 then slumpgurka      // 50% sannolikhet för gurka
  else if rnd > 0.2 then slumptomat // 30% sannolikhet för tomat
  else slumpbroccoli             // 20% sannolikhet för broccoli
```

c) Fördelarna med `match`-versionen, och mönstermatchning i sig, är att det är väldigt lätt att göra ändringar på hur matchningen sker. Detta innebär att det skulle vara väldigt lätt att ändra definitionen för ätbarheten. Skulle dock dessa inte ändras ofta utan snarare grönsaksutbudet så kan det polyformistiska alternativet vara att föredra. Detta eftersom det skulle implementeras och ändras lättare än mönstermatchningen vid byte av grönsaker.

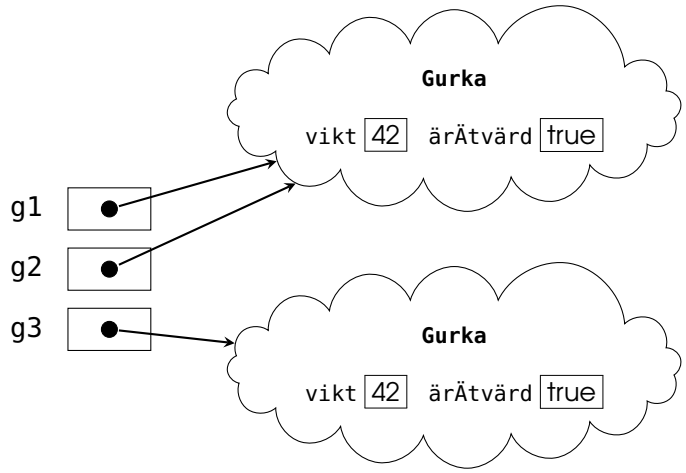
Lösn. uppg. 12. *Metoden* `equals`.

a)

1. En klass `Gurka` skapas med parametrarna `vikt` av typen `Int` och ärÄtbar av typen `Boolean`.
2. `g1` tilldelas en instans av `Gurka`-klassen med `vikt = 42` och `ärÄtbar = true`.
3. `g2` tilldelas samma `Gurka`-objekt som g1.
4. `g3` tilldelas en ny instans av `Gurka`-klassen med motsvarande parametrar som g1.
5. ==(equals)-metoden jämför g1 med g2 och returnerar `true`.
6. ==(equals)-metoden jämför g1 med g3 och returnerar `false`.
7. `def equals(x\$1: Any): Boolean`

Som kan ses ovan är elementet som jämförs i `equals` av typen `Any`. Eftersom programmet inte känner till klassen så används `Any.equals` vid jämförelsen. Till skillnad från de primitiva datatyperna som vid jämförelse med `equals` jämför innehållslikhet, så jämförs referenslikheten hos klasser om inget annat är specificerat. `g1` och `g2` refererar till samma objekt medan `g3` pekar på ett eget sådant vilket innebär att `g1` och `g3` inte har referenslikhet.

b)



c) -

d) I de första 3 raderna sker samma som i deluppgift *a*. När nu dessa jämförelser görs mellan `Gurka`-objekten så överskuggas `Any.equals` av den `equals` som är specificerad för just `Gurka`. Eftersom båda objekten `g1` jämförs med också är av typen `Gurka` så matchar den med `case that: Gurka`. Denna i sin tur jämför vikterna hos de båda gurkorna och returnerar en `Boolean` huruvida de är lika eller inte, vilket de i båda fallen är.

e) I deluppgift a gav `g1 == g3` `false` trots innehållslikhet. Efter skuggningen ger dock detta uttryck `true` vilket påvisar jämförelse av innehållslikhet.

Lösn. uppg. 13. *Polynom.*

a) TODO!!!

b) TODO!!!

Lösn. uppg. 14. *`Option` som en samling.* TODO!!!

Lösn. uppg. 15. *Fånga undantag med* **`catch`** *i Java och Scala.* TODO!!!

Lösn. uppg. 16. *Polynom, fortsättning: reducering.*

Lösn. uppg. 17. *Typsäker innehållstest med metoden* `===`.

Lösn. uppg. 18. *Överskugga* `equals` *med innehållslikhet även för icke-finala klasser.*

Lösn. uppg. 19. *Överskugga equals vid arv.*

Lösn. uppg. 20. *Speciella matchningar.* TODO!!!

Lösn. uppg. 21. *Extraktorer.* TODO!!!

Lösn. uppg. 22. *Polynom, fortsättning: polynomdivision.* TODO!!!

7. Lösning sequences

7.1 Grunduppgifter; förberedelse inför laboration

Lösn. uppg. 1. *Para ihop begrepp med beskrivning.*

element	1	⇝	G	objekt i en datastruktur
samling	2	⇝	B	datastruktur med element av samma typ
samlingsbibliotek	3	⇝	J	många färdiga samlingar med olika egenskaper
sekvens(samling)	4	⇝	L	noll el. flera element av samma typ i viss ordning
sekvensalgoritm	5	⇝	I	lösning på problem som drar nytta av sekvenssamling
ordning	6	⇝	A	definierar hur element av en viss typ ska ordnas
sortering	7	⇝	C	algoritm som ordnar element i en viss ordning
sökning	8	⇝	D	algoritm som letar upp element enligt sökkriterium
linjärsökning	9	⇝	F	sökalgoritm som letar i sekvens tills element hittas
registrering	10	⇝	H	algoritm som räknar element med vissa egenskaper
tidskomplexitet	11	⇝	E	hur exekveringstiden växer med problemstorleken
minneskomplexitet	12	⇝	K	hur minnesåtgången växer med problemstorleken

Lösn. uppg. 2. *Olika sekvenssamlingar.*

`Vector`	1	⇝	B	oföränderlig, ger snabbt godtyckligt ändrad samling
`List`	2	⇝	C	oföränderlig, ger snabbt ny samling ändrad i början
`Array`	3	⇝	D	primitiv, förändringsbar, snabb indexering, fix storlek
`ArrayBuffer`	4	⇝	A	förändringsbar, snabb indexering, kan ändra storlek
`ListBuffer`	5	⇝	E	förändringsbar, snabb att ändra i början

Lösn. uppg. 3. *Använda sekvenssamlingar.*

a)

`x +: xs`	1	⇝ G	`Vector(0, 1, 2, 3)`
`xs +: x`	2	⇝ L	`error: value +: is not a member of Int`
`xs :+ x`	3	⇝ J	`Vector(1, 2, 3, 0)`
`xs ++ xs`	4	⇝ M	`Vector(1, 2, 3, 1, 2, 3)`
`xs.indices`	5	⇝ E	`(0 until 3)`
`xs apply 0`	6	⇝ C	`1`
`xs(3)`	7	⇝ I	`java.lang.IndexOutOfBoundsException`
`xs.length`	8	⇝ O	`3`
`xs.take(4)`	9	⇝ F	`Vector(1, 2, 3)`
`xs.drop(2)`	10	⇝ K	`Vector(3)`
`xs.updated(0, 2)`	11	⇝ B	`Vector(2, 2, 3)`
`xs.tail.head`	12	⇝ N	`2`
`xs.head.tail`	13	⇝ D	`error: value tail is not a member of Int`
`xs.isEmpty`	14	⇝ H	`false`
`xs.nonEmpty`	15	⇝ A	`true`

b)

fel	*typ*	*förklaring*
`value +: is not a member of Int`	kompileringsfel	Operatorer som slutar med kolon är högerassociativa. Metodanropet `xs +: x` motsvarar med punktnotation `x.+:(xs)` och det finns ingen metod med namnet `+:` på heltal.
`IndexOutOfBoundsException`	körtidsfel	Det finns bara 3 element och index räknas från 0 i sekvenssamlingar.
`value tail is not a member of Int`	kompileringsfel	Metoden `head` ger första elementet och heltal saknar sekvenssamlingsmetoden `tail`.

Lösn. uppg. 4. *Kopiering av sekvenser.*

a)

`xs(0)`	`rs$line5$Mutant@66d766b9` nya instanser får nya hexkoder
`ys(0).int`	`0` eftersom `ys` innehåller samma instans som `xs`
`zs(0).int`	`5` eftersom `!(xs(0) eq zs(0))`
`xs(0) eq ys(0)`	`true` eftersom samma instans
`xs(0) eq zs(0)`	`false` eftersom olika instanser
`(ys.toBuffer :+ new Mutant).apply(0).int`	`0` eftersom den ej djupkopierade kopian av typen `ArrayBuffer` refererar samma instans på första platsen som både `ys` och `xs` och `x(0).int` blev noll i en tilldelning på rad 5 i REPL-körningen

Observera alltså att kopiering med `toArray`, `toVector`, `toBuffer`, etc. *inte är djup*, d.v.s. det är bara instansreferenserna som kopieras och inte själva instanserna.

b)

```
def deepCopy(xs: Array[Mutant]): Array[Mutant] =
  val result = Array.ofDim[Mutant](xs.length) //fylld med null-referenser
  var i = 0
  while i < xs.length do
    result(i) = new Mutant(xs(i).int) //kopia med samma innehåll på samma plats
    i += 1
  result
```

Det går också bra att skapa resultatarrayen med `new Array[Mutant](xs.length)`. Du kan också använda `size` i stället för `length`.

c)

```
scala> class Mutant(var int: Int = 0)
// defined class Mutant

scala> def deepCopy(xs: Array[Mutant]): Array[Mutant] =
     |   val result = Array.ofDim[Mutant](xs.length)
     |   var i = 0
     |   while i < xs.length do
     |     result(i) = new Mutant(xs(i).int)
     |     i += 1
     |   result

scala> val xs = Array.fill(3)(new Mutant)
xs: Array[Mutant] = Array(rs$line$2$Mutant@46a123e4, rs$line$2$Mutant@44bc2449,
rs$line2$Mutant@3c28e5b6)

scala> val ys = deepCopy(xs)
ys: Array[Mutant] = Array(rs$line$2$Mutant@14b8a751, rs$line2$Mutant@7345f97d,
rs$line$2$Mutant@554566a8)

scala> xs(0).int = 5

scala> ys(0).int
val res0: Int = 0
```

d) Nej, eftersom elementen inte kan förändras kan man utan problem dela referenser mellan samlingar. Det finns inte någon möjlighet att det kan ske förändringar som påverkar flera samlingar samtidigt. Dock gör man vanligen (ofta tidsödande) djupkopieringar av samlingar med förändringsbara element för att kunna vara säker på att den ursprungliga samlingen inte förändras.

Lösn. uppg. 5. *Uppdatering av sekvenser.*

a)

`{ buf(0) = -1; buf(0) }`	1	⇝ D	`-1`
`{ xs(0) = -1; xs(0) }`	2	⇝ A	`error: value update is not a member`
`buf.update(1, 5)`	3	⇝ F	`(): Unit`
`xs.updated(0, 5)`	4	⇝ B	`Vector(5, 2, 3, 4)`
`{ buf += 5; buf }`	5	⇝ C	`ArrayBuffer(-1, 5, 3, 4, 5)`
`{ xs += 5; xs }`	6	⇝ G	`error: value += is not a member`
`xs.patch(1,Vector(-1,5),3)`	7	⇝ E	`Vector(1, -1, 5)`
`xs`	8	⇝ H	`Vector(1, 2, 3, 4)`

b)

```
def insert(xs: Array[Int], elem: Int, pos: Int): Array[Int] =
  xs.patch(from = pos, other = Array(elem), replaced = 0)
```

c) Pseudokoden nedan är skriven så att den kompilerar fast den är ofärdig.

```
def insert(xs: Array[Int], elem: Int, pos: Int): Array[Int] =
  val result = ??? /* ny array med plats för ett element mer än i xs */
  var i = 0
  while(???){/* kopiera elementen före plats pos och öka i */}
  if i < result.length then /* lägg elem i result på plats i */
  while(???){/* kopiera över resten */}
  result
```

d)

```
def insert(xs: Array[Int], elem: Int, pos: Int): Array[Int] =
  val result = new Array[Int](xs.length + 1)
  var i = 0
  while i < pos && i < xs.length do  { result(i) = xs(i); i += 1}
  if i < result.length then { result(i) = elem; i += 1 }
  while i < result.length do { result(i) = xs(i - 1); i += 1}
  result
```

```
scala> insert(Array(1, 2), 0, pos = -1)
val res2: Array[Int] = Array(0, 1, 2)

scala> insert(Array(1, 2), 0, pos = 0)
val res3: Array[Int] = Array(0, 1, 2)

scala> insert(Array(1, 2), 0, pos = 1)
val res4: Array[Int] = Array(1, 0, 2)

scala> insert(Array(1, 2), 0, pos = 2)
val res5: Array[Int] = Array(1, 2, 0)

scala> insert(Array(1, 2), 0, pos = 42)
val res7: Array[Int] = Array(1, 2, 0)
```

Lösn. uppg. 6. *Jämföra strängar i Scala.*

a)

```
true
true
true
true
true
false
```

b) *s1* kommer först.

Lösn. uppg. 7. *Linjärsökning enligt olika sökkriterier.*

a)

`xs.indexOf(0)`	1	⇝ F	`5`
`xs.indexOf(6)`	2	⇝ B	`-1`
`xs.indexWhere(_ < 2)`	3	⇝ H	`4`
`xs.indexWhere(_ != 5)`	4	⇝ I	`1`
`xs.find(_ == 1)`	5	⇝ D	`Some(1)`
`xs.find(_ == 6)`	6	⇝ J	`None`
`xs.contains(0)`	7	⇝ C	**`true`**
`xs.filter(_ == 1)`	8	⇝ A	`Vector(1, 1)`
`xs.filterNot(_ > 1)`	9	⇝ E	`Vector(1, 0, 1)`
`xs.zipWithIndex.filter(_._1 == 1).map(_._2)`	10	⇝ G	`Vector(4, 6)`

b) Med en boolesk variabel `found`:

```
def indexOf(xs: Vector[String], p: String => Boolean): Int =
  var found = false
  var i = 0
  while i < xs.length && !found do
      found = p(xs(i))
      i += 1
  if found then i - 1 else -1
```

Eller utan `found`:

```
def indexOf(xs: Vector[String], p: String => Boolean): Int =
  var i = 0
  while i < xs.length && !p(xs(i)) do i += 1
  if i == xs.length then -1 else i
```

Eller så kanske man vill börja bakifrån; lösningen nedan är nog enklare att fatta (?) och definitivt mer koncis, men uppfyller *inte* kravet att returnera index för *första* förekomsten som det står i uppgiften. Men om sammanhanget tillåter att vi returnerar *något* index för vilket predikatet gäller, eller om man faktiskt har kravet att leta bakifrån, så funkar detta:

```
def indexOf(xs: Vector[String], p: String => Boolean): Int =
```

```
  var i = xs.length - 1
  while i >= 0 && !p(xs(i)) do i -= 1
  i
```

Eller så kan man göra på flera andra sätt. När du ska implementera algoritmer, både på programmeringstentan och i yrkeslivet som systemutvecklare, finns det ofta många olika sätt att lösa uppgiften på som har olika egenskaper, fördelar och nackdelar. Det viktiga är att lösningen fungerar så gott det går enligt kraven, att koden är begriplig för människor och att implementationen inte är så ineffektiv att användarna tröttnar i sin väntan på resultatet...

Lösn. uppg. 8. *Labbförberedelse: Implementera heltalsregistrering i Array.*

a)

```
def registreraTärningskast(xs: Seq[Int]): Vector[Int] =
  val result = Array.fill(6)(0)
  xs.foreach{ x =>
    require(x >= 1 && x <= 6, "tärningskast ska vara mellan 1 & 6")
    result(x - 1) += 1
  }
  result.toVector
```

b)

```
scala> registreraTärningskast(kasta(1000))
val res0: Vector[Int] = Vector(171, 163, 166, 152, 184, 164)

scala> registreraTärningskast(kasta(1000))
val res1: Vector[Int] = Vector(163, 161, 158, 174, 161, 183)
```

Lösn. uppg. 9. *Inbyggda metoder för sortering.*

`'a' < 'A'`	1	⇝ E	`false`
`"AÄÖö" < "AÅÖö"`	2	⇝ H	`true`
`xs.sorted.head`	3	⇝ C	`-1`
`xs.sorted.reverse.head`	4	⇝ G	`3`
`ys.sorted.head`	5	⇝ I	`"ak"`
`zs.indexOf('a')`	6	⇝ B	`1`
`ps.sorted.head.förnamn.take(2)`	7	⇝ D	`error: ...`
`ps.sortBy(_.förnamn).apply(1).förnamn.take(2)`	8	⇝ A	`"ka"`
`xs.sortWith((x1,x2) => x1 > x2).indexOf(3)`	9	⇝ F	`0`

Det blir fel i uttrycket ovan som försöker sortera en sekvens med instanser av `Person` direkt med metoden `sorted`:

```
scala> ps.sorted
No implicit Ordering defined for Person.
```

Det blir fel eftersom kompilatorn inte hittar någon ordningsdefinition för dina egna klasser. Senare i kursen ska vi se hur vi kan skapa egna ordningar om man vill få

`sorted` att fungera på sekvenser med instanser av egna klasser, men ofta räcker det fint med `sortBy` och `sortWith`.

Lösn. uppg. 10. *Inbyggd metod för blandning.*

a) `Random.shuffle` returnerar en ny blandad sekvenssamling av samma typ. Ordningen i den ursprungliga samlingen påverkas inte.

b) Exempel på användning av `random.shuffle`:

```
scala> import scala.util.Random

scala> val xs = Vector("Sten", "Sax", "Påse")
val xs: Vector[String] = Vector(Sten, Sax, Påse)

scala> (1 to 10).foreach(_ => println(Random.shuffle(xs).mkString(" ")))
Sax Påse Sten
Sten Påse Sax
Sten Sax Påse
Sten Sax Påse
Sten Påse Sax
Sten Påse Sax
Sax Sten Påse
Sten Påse Sax
Sax Påse Sten
Sax Påse Sten

scala> (1 to 5).map(_ => Random.shuffle(1 to 6))
val res1: IndexedSeq[IndexedSeq[Int]] =
  Vector(Vector(5, 2, 1, 4, 3, 6), Vector(6, 5, 4, 2, 1, 3),
  Vector(3, 1, 4, 6, 5, 2), Vector(3, 2, 6, 5, 1, 4),
  Vector(5, 3, 4, 6, 1, 2))

scala> (1 to 1000).map(_ => Random.shuffle(1 to 6).head).count(_ == 6)
val res2: Int = 168
```

Lösn. uppg. 11. *Repeterade parametrar.*

a)

```
scala> def stringSizes(xs: String*): Vector[Int] = xs.map(_.size).toVector
def stringSizes(xs: String*): Vector[Int]

scala> stringSizes("hej")
val res0: Vector[Int] = Vector(3)

scala> stringSizes("hej", "på", "dej", "")
val res1: Vector[Int] = Vector(3, 2, 3, 0)

scala> stringSizes()
val res2: Vector[Int] = Vector()
```

Anrop med tom argumentlista ger en tom heltalssekvens.

b)

```
scala> val xs = Vector("hej","på","dej", "")
val xs: Vector[String] = Vector(hej, på, dej, "")

```

```
scala> stringSizes(xs: _*)
val res0: Vector[Int] = Vector(3, 2, 3, 0)

scala> stringSizes(Vector(): _*)
val res1: Vector[Int] = Vector()
```

Ja, det funkar fint med tom sekvens.

7.2 Extrauppgifter; träna mer

Lösn. uppg. 12. *Registrering av booleska värden. Singla slant.*

a)

```
def registerCoinFlips(xs: Seq[Boolean]): (Int, Int) =
  val result = Array.fill(2)(0)
  xs.foreach(x => if (x) result(0) += 1 else result(1) += 1)
  (result(0), result(1))
```

b)

Lösn. uppg. 13. *Kopiering och tillägg på slutet.*

```
def copyAppend(xs: Array[Int], x: Int): Array[Int] =
  val ys = new Array[Int](xs.length + 1)
  var i = 0
  while i < xs.length do
    ys(i) = xs(i)
    i += 1
  ys(xs.length) = x
  ys
```

De två buggarna i algoritmen finns (1) i villkoret som ska vara strikt mindre än och (2) inne i loopen där uppräkningen av loppvariabeln saknas.

Lösn. uppg. 14. *Kopiera och reversera sekvens.*

a)

```
def seqReverseCopy(xs: Array[Int]): Array[Int] =
  val n = xs.length
  val ys = new Array[Int](n)
  var i = 0
  while i < n do
    ys(n - i - 1) = xs(i)
    i += 1
  ys
```

b)

```
def seqReverseCopy(xs: Array[Int]): Array[Int] =
  val n = xs.length
  val ys = new Array[Int](n)
  for i <- (n - 1) to 0 by -1 do
    ys(n - i - 1) = xs(i)
  ys
```

Lösn. uppg. 15. *Kopiera alla utom ett.*

Indata : En sekvens xs av typen `Array[Int]` och pos
Utdata : En ny sekvens av typen `Array[Int]` som är en kopia av xs fast med elementet på plats pos borttaget

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** $i \leftarrow pos+1$ **to** $n-1$ **do**
8 $\quad ys(i-1) \leftarrow xs(i)$
9 **end**
10 ys

```
def removeCopy(xs: Array[Int], pos: Int): Array[Int] =
  val n = xs.size
  val ys = Array.fill(n - 1)(0)
  for i <- 0 until pos do
    ys(i) = xs(i)
  for i <- (pos + 1) until n do
    ys(i - 1) = xs(i)
  ys
```

Lösn. uppg. 16. *Borttagning på plats i array.*

Indata : En sekvens xs av typen `Array[Int]`, en position pos och ett utfyllnadsvärde pad
Utdata : En uppdaterad sekvens av xs där elementet på plats pos tagits bort och efterföljande element flyttas ett steg mot lägre index med ett sista elementet som tilldelats värdet av pad

1 $n \leftarrow$ antalet element xs
2 **for** $i \leftarrow pos+1$ **to** $n-1$ **do**
3 $\quad xs(i-1) \leftarrow xs(i)$
4 **end**
5 $xs(n-1) \leftarrow pad$

```
def remove(xs: Array[Int], pos: Int, pad: Int = 0): Unit =
  val n = xs.size
  for i <- (pos + 1) until n do
    xs(i - 1) = xs(i)
  xs(n - 1) = pad
```

Lösn. uppg. 17. *Kopiering och insättning.*

a)

```
def insertCopy(xs: Array[Int], x: Int, pos: Int): Array[Int] =
  val n = xs.size
  val ys = Array.ofDim[Int](n + 1)
  for i <- 0 until pos do
    ys(i) = xs(i)
```

```
  ys(pos) = x
  for i <- pos until n do
    ys(i + 1) = xs(i)
  ys
```

b) `pos` måste vara `0`.

c)

```
java.lang.ArrayIndexOutOfBoundsException: -1
```

d) Elementet `x` läggs till på slutet av arrayen, alltså kommer den returnerande arrayen vara större än den som skickades in.

e)

```
java.lang.ArrayIndexOutOfBoundsException: 5
```

Man får `ArrayIndexOutOfBoundsException` då indexeringen är utanför storleken hos arrayen.

Lösn. uppg. 18. *Insättning på plats i array.*

Indata: En sekvens xs av typen `Array[Int]` och heltalen x och pos
Utdata: xs uppdaterat på plats, där elementet x har satts in på platsen pos och efterföljande element flyttas ett steg där sista elementet försvinner

1 $n \leftarrow$ antalet element i xs
2 $ys \leftarrow$ en klon av xs
3 $xs(pos) \leftarrow x$
4 **for** $i \leftarrow pos + 1$ **to** $n - 1$ **do**
5 $\quad xs(i) \leftarrow ys(i - 1)$
6 **end**

```
def insertDropLast(xs: Array[Int], x: Int, pos: Int): Unit =
  val n = xs.size
  val ys = xs.clone
  xs(pos) = x
  for i <- pos + 1 until n do
    xs(i) = ys(i - 1)
```

Lösn. uppg. 19. *Fler inbyggda metoder för linjärsökning.*

a)

- `lastIndexOf` är bra om man vill leta bakifrån i stället för framifrån; utan denna hade man annars då behövt använda `xs.reverse.indexOf(e)`
- `indexOfSlice(ys)` letar efter index där en hel sekvens `ys` börjar, till skillnad från `indexOf(e)` som bara letar efter ett enstaka element.
- `segmentLength(p, i)` ger längden på den längsta sammanhängande sekvens där alla element uppfyller predikatet `p` och sökningen efter en sådan sekvens börjar på plats `i`
- `xs.maxBy(f)` kör först funktionen `f` på alla element i `xs` och letar sedan upp det största värdet; motsvarande `minBy(f)` ger minimum av `f(e)` över alla element `e` i `xs`

b) –

7.3 Fördjupningsuppgifter; utmaningar

Lösn. uppg. 20. *Fixa svensk sorteringsordning av ÄÅÖ.*

Lösn. uppg. 21. *Fibonacci-sekvens med ListBuffer.*

a)

```
def fib(max: Long): List[Long] =
  val xs = scala.collection.mutable.ListBuffer.empty[Long]
  xs.prependAll(Vector(1, 1))
  while xs.head < max do xs.prepend(xs.take(2).sum)
  xs.reverse.drop(1).toList
```

b)

```
scala> fib(Int.MaxValue).size
val res0: Int = 46
```

c)

```
def fibBig(max: BigInt): List[BigInt] =
  val xs = scala.collection.mutable.ListBuffer.empty[BigInt]
  xs.prependAll(Vector(BigInt(1), BigInt(1)))
  while xs.head < max do xs.prepend(xs.take(2).sum)
  xs.reverse.drop(1).toList
```

```
scala> fibBig(Long.MaxValue).size
val res0: Int = 92

scala> fibBig(BigInt(Long.MaxValue).pow(64)).size
val res1: Int = 5809

scala> fibBig(BigInt(Long.MaxValue).pow(128)).last
val res2: BigInt = 46657280552835544919455361110286315395072000518604554717752
5

scala> fibBig(BigInt(Long.MaxValue).pow(128)).last.toString.size
val res3: Int = 2428

scala> fibBig(BigInt(Long.MaxValue).pow(256)).last.toString.size
val res4: Int = 4856

scala> fibBig(BigInt(Long.MaxValue).pow(1024)).last.toString.size
java.lang.OutOfMemoryError: Java heap space
```

Lösn. uppg. 22. *Omvända sekvens på plats.*

```
def reverseChars(xs: Array[Char]): Unit =
  val n = xs.length
  for i <- 0 to (n/2 - 1) do
    val temp = xs(i)
    xs(i) = xs(n - i - 1)
    xs(n - i - 1) = temp
```

Lösn. uppg. 23. *Palindrompredikat.*

a) Omvändning med `reverse` kan kräva genomgång av hela strängen en gång samt minnesutrymme för kopian. Innehållstestet kräver ytterligare en genomgång. (Detta är i och för sig inget stort problem eftersom världens längsta palindrom inte är längre än 19 bokstäver och är ett obskyrt finskt ord som inte ofta yttras i dagligt tal. Vilket?)

b)

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

Lösn. uppg. 24. *Fler användbara sekvenssamlingsmetoder.*

```
scala> val xs = Vector.tabulate(10)(i => math.pow(2, i).toInt)
xs: Vector[Int] = Vector(1, 2, 4, 8, 16, 32, 64, 128, 256, 512)

scala> xs.forall(_ < 1024)
val res0: Boolean = true

scala> xs.exists(_ == 3)
val res1: Boolean = false

scala> xs.count(_ > 64)
val res2: Int = 3

scala> xs.zipWithIndex.take(5)
val res3: Vector[(Int, Int)] = Vector((1,0), (2,1), (4,2), (8,3), (16,4))
```

Lösn. uppg. 25. *Arrays don't behave, but `ArraySeqs` do!*

a) `xs` erbjuder innehållslikhet och har typen `Seq[Int]` med den underliggande typen `ArraySeq[Int]`. Det går inte att göra tilldelning av element i en `ArraySeq` eftersom metoden `update` saknas, och den är oföränderlig. Den uppdateras därför inte när den ursprungliga arrayen uppdateras.

```
scala> val as1 = Array(1,2,3)
val as1: Array[Int] = Array(1, 2, 3)

scala> val as2 = Array(1,2,3)
val as2: Array[Int] = Array(1, 2, 3)


scala> val (xs1, xs2) = (as1.toSeq, as2.toSeq)
val xs1: Seq[Int] = ArraySeq(1, 2, 3)
val xs2: Seq[Int] = ArraySeq(1, 2, 3)

scala> as1 == as2
val res0: Boolean = false

```

```
scala> xs1 == xs2
val res1: Boolean = true

scala> as1(0) = 42

scala> xs1
val res2: Seq[Int] = ArraySeq(1, 2, 3)

scala> xs1(0) = 42
value update is not a member of Seq[Int]
```

b) Vid repeterade parametrar får man en `ArraySeq`.

```
scala> def f(xs: Int*) = xs
def f(xs: Int*): Seq[Int]

scala> println(f(1,2,3))
ArraySeq(1, 2, 3)
```

c) Det går inte att ha en generisk array som funktionsresultat utan att bifoga kontextgränsen `ClassTag` i typparametern för att kompilatorn ska kunna generera kod för den typkonvertering som krävs under runtime av JVM. Se exempel här: http://docs.scala-lang.org/overviews/collections/arrays.html

Lösn. uppg. 26. Sekvenssamlingen `List` är nästan dubbelt så snabb vid bearbetning i början men ungefär 1000 gånger långsammare vid bearbetning i slutet av en sekvens med 100000 element.

Olika körningar går olika snabbt på JVM bl.a. p.g.a optimeringar som sker när JVM-en ”värms upp” och den så kallade Just-In-Time-kompileringen gör sitt mäktiga jobb. Det går ibland plötsligt väsentligt långsammare när skräpsamlaren tvingas göra tidsödande storstädning av minnet.

Lösn. uppg. 27. –

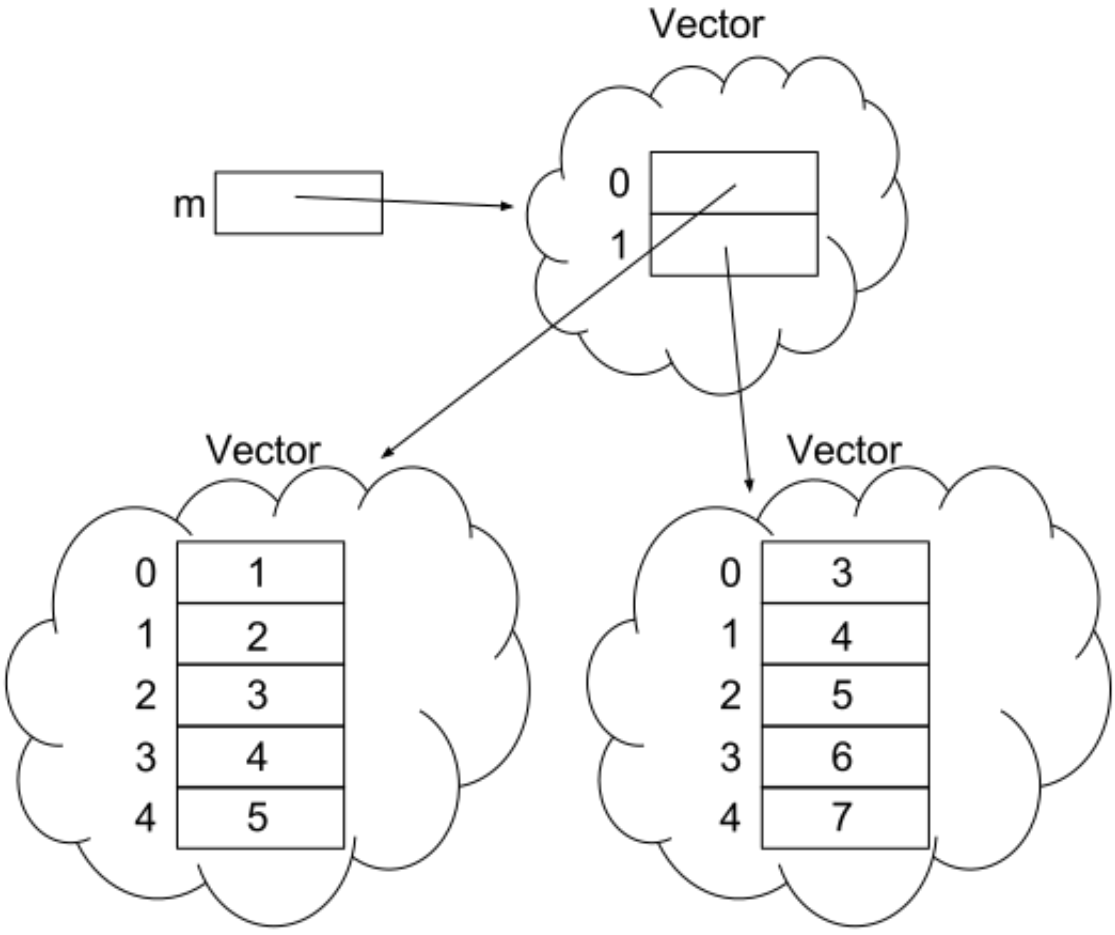
8. Lösning matrices

8.1 Grunduppgifter; förberedelse inför laboration

Lösn. uppg. 1. *Para ihop begrepp med beskrivning.*

matris	1	⇝	A	indexerbar datastruktur i två dimensioner
radvektor	2	⇝	F	matris av dimension $1 \times m$ med m horisontella värden
kolumnvektor	3	⇝	G	matris av dimension $m \times 1$ med m vertikala värden
kolonn	4	⇝	C	annat ord för kolumn
generisk	5	⇝	B	har abstrakt typparameter, typen är generell
typargument	6	⇝	D	konkret typ, binds till typparameter vid kompilering
typhärledning	7	⇝	E	kompilatorn beräknar typ ur sammanhanget

Lösn. uppg. 2. *Skapa matriser med hjälp av nästlade samlingar.*



a)
Typ: `Vector[Vector[Int]]`
Värde: `Vector(Vector(1, 2, 3, 4, 5), Vector(3, 4, 5, 6, 7))`
Dimensioner: 2×5
Inom matematiken sker indexering enligt konvention med 1 som lägsta index. I scala

är lägsta index 0, man använder s.k. 0-indexering. [3]

b)

```
scala> val m = Vector((1 to 5).toVector, (3 to 7).toVector)
m: Vector[Vector[Int]] = Vector(Vector(1, 2, 3, 4, 5), Vector(3, 4, 5, 6, 7))

scala> m.apply(0).apply(1)
res4: Int = 2

scala> m(1)
res5: Vector[Int] = Vector(3, 4, 5, 6, 7)

scala> m(1)(4)
res6: Int = 7
```

c)
m2: `Vector[Vector[Int]]`
m3: `Vector[Vector[AnyVal]]`
m4: `Vector[Vector[Any]]`
m5: `Vector[Vector[Int]]`

d) m5, 42×2

Lösn. uppg. 3. *Skapa och iterera över matriser.*

a)

```
def throwDie: Int = (math.random() * 6).toInt + 1
```

Eller:

```
def throwDie: Int = scala.util.Random.nextInt(6) + 1
```

b) Matrisdimension i matematisk notation: 1000×5, vilket motsvarar en matris med 1000 rader och 5 kolumner.

c)

```
ds1: IndexedSeq[IndexedSeq[Int]]
ds2: IndexedSeq[IndexedSeq[Int]]
ds3: IndexedSeq[Vector[Int]]
ds4: IndexedSeq[Vector[Int]]
ds5: Vector[Vector[Int]]
ds6: Vector[Vector[Int]]
```

`IndexedSeq` och `Vector` ovan finns i paketet `scala.collection.immutable`

d)

```
def roll(n: Int) = Vector.fill(n)(throwDie).sorted
```

e)

```
def isYatzy(xs: Vector[Int]): Boolean = xs.forall(_ == xs(0))
```

[3] Detta är inte fallet i alla programmeringsspråk, vilket du kan läsa mer om på https://en.wikipedia.org/wiki/Array_data_type#Index_origin

f)

```
def diceMatrix(m: Int, n: Int): Vector[Vector[Int]] =
  Vector.fill(m)(roll(n))
```

g)

```
def diceMatrixToString(xss: Vector[Vector[Int]]): String =
  xss.map(_.mkString(" ")).mkString("\n")
```

h)

```
def filterYatzy(xss: Vector[Vector[Int]]): Vector[Vector[Int]] =
  xss.filter(isYatzy)
```

i)

```
def yatzyPips(xss: Vector[Vector[Int]]): Vector[Int] =
  filterYatzy(xss).map(_.head)
```

Lösn. uppg. 4. *En oföränderlig, generisk matris-klass till veckans laboration* `life`.

a) Typen på `m` blir `Matrix`.

b) Typen på `e` blir `String`.

c) Man behöver ändra på 3 ställen från `String` till `Int`.

d) Generisk matris `Matrix[T]` för element av godtycklig typ `T`:

```
case class Matrix[T](data: Vector[Vector[T]]):
  def apply(row: Int, col: Int): T = data(row)(col)

object Matrix:
  def fill[T](dim: (Int, Int))(value: T): Matrix[T] =
    Matrix[T](Vector.fill(dim._1, dim._2)(value))
```

e) Tack vare kompilatorns typinferens så får `bm` typen `Matrix[Boolean]`.

f) Typen på `be` blir `Boolean`.

g) h) i) j) k) är alla implementerade i koden nedan:

```
case class Matrix[T](data: Vector[Vector[T]]):
  require(data.forall(row => row.length == data(0).length))

  val dim: (Int, Int) = (data.length, data(0).length)

  def apply(row: Int, col: Int): T = data(row)(col)

  def updated(row: Int, col: Int)(value: T): Matrix[T] =
    Matrix(data.updated(row, data(row).updated(col, value)))

  def foreachIndex(f: (Int, Int) => Unit): Unit =
    for r <- data.indices; c <- data(r).indices do f(r, c)

  override def toString =
    s"""Matrix of dim $dim:\n${ data.map(_.mkString(" ")).mkString("\n") }"""
```

```
object Matrix:
  def fill[T](dim: (Int, Int))(value: T): Matrix[T] =
    Matrix[T](Vector.fill(dim._1, dim._2)(value))
```

8.2 Extrauppgifter; träna mer

Lösn. uppg. 5. *Imperativa matrisalgoritmer.*

a)

```
def isYatzy(xs: Vector[Int]): Boolean =
  var foundDiff = false
  var i = 0
  while (i < xs.size && !foundDiff) do
    foundDiff = xs(i) != xs(0)
    i += 1
  end while
  !foundDiff
```

b) Funktionen går igenom varje matrisrad, där den i sin tur går igenom varje element på raden och lägger till i `StringBuilder`-objektet. Om det inte är det sista elementet på raden läggs även ett blanktecken till, annars läggs ett nyradstecken till. Undantaget är sista raden, där inget nyradstecken läggs till. Slutligen konverteras `StringBuilder`-objektet till en `String` som returneras.

Är `xss` tom blir `xss.indices` en tom `Range` och den yttre `for`-loopen hoppas över och en tom sträng returneras. Är alla rader tomma hoppas i stället de inre `for`-looparna över, med samma resultat.

Fördel: `StringBuilder` är snabbare vid tillägg på slutet vid stora strängar (men här kommer det inte märkas eftersom strängen är så liten).

Nackdel: StringBuilder-koden uppfattas av många som svårare att läsa.

c)

```
def filterYatzy(xss: Vector[Vector[Int]]): Vector[Vector[Int]] =
  var result: Vector[Vector[Int]] = Vector()
  for i <- xss.indices if isYatzy(xss(i)) do result = result :+ xss(i)
  result
```

d) Varje looprunda ger en vektor `xss(i)` om filtervillkoret är uppfyllt och resultatet av `for`-uttrycket blir en vektor med vektorer som är yatzyslag.

Lösn. uppg. 6. *Strängtabell med kolumnrubriker.*

a)

```
case class Table(
  data: Vector[Vector[String]],
  headings: Vector[String],
  sep: Char
):

  val dim: (Int, Int) = (data.size, headings.size)

  def apply(r: Int, c: Int): String = data(r)(c)

  def row(r: Int): Vector[String]= data(r)

  def col(c: Int): Vector[String] = data.map(r => r(c))
```

```
  lazy val indexOfHeading: Map[String, Int] = headings.zipWithIndex.toMap

  def col(h: String): Vector[String] = col(indexOfHeading(h))

  def values(h: String): Vector[String] = col(h).distinct.sorted

  override def toString: String =
    val s = sep.toString
    headings.mkString(s) + "\n" +data.map(_.mkString(s)).mkString("\n")

object Table:
  def fromFile(fileName: String, sep: Char = ';'): Table =
    val lines = scala.io.Source.fromFile(fileName).getLines.toVector
    val matrix= lines.map(_.split(sep).toVector)
    new Table(matrix.tail, matrix.head, sep)
```

b)

```
@main
def run(fileName: String, separator: String): Unit =
  require(separator.length == 1, "separator ska vara exakt ett tecken")
  val t = Table.fromFile(fileName, separator.head)
  val counts: Vector[Vector[String]] =
    (0 until t.dim._2)
      .map(i => t.values(t.headings(i))
      .map(x => s"$x: ${t.col(i).count(_ == x)}"))
      .toVector
  for (i <- 0 until t.dim._2) do
    println(s"\nColumn: ${i + 1}, ${t.headings(i)}:")
    for (j <- 0 until counts(i).length) do
      println(counts(i)(j))
```

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

–

8.3 Fördjupningsuppgifter; utmaningar

Lösn. uppg. 8. *Generiska funktioner.*

a)

1. –
2. Strängrepresentationen av `42` spegelvänds
3. `"hej"` spegelvänds - `toString` av en sträng ger en likadan sträng
4. –
5. Gurk-objektets strängrepresentation spegelvänds
6. Funktionens typparameter matchar inte parameterns typ: `42` är ingen sträng
7. Implicit typkonvertering till `Double` sker för att stämma överens med typparametern, vilket ger en strängrepresentation med decimal

b)

1. En funktion definieras så att den tar emot två andra funktioner som argument, sätter ihop dem, och matar in ett tredje argument till den den sammansatta funktionen.
2. En funktion som inkrementerar ett heltal med 1 definieras.
3. En funktion som halverar ett flyttal definieras.
4. `42` matas in i `inc()` och resultatet (`43`) matas vidare till `half()`. Inuti `half()` sker implicit typkonvertering till `Double` då talet divideras med ett flyttal (`2.0`) och resultatet blir `43.0 / 2.0`, alltså `21.5`.
5. Resultatet från `half()` är av typ `Double`, medan `inc()` tar emot ett argument av typ `Int`. Då flyttal generellt inte kan konverteras till heltal utan informationsförlust sker ingen implicit konvertering, istället sker ett kompileringsfel.

c)

```
def inc(x: Double): Double = x + 1.0
```

Nu ges kompileringsfel på rad 4 istället, vilket kan lösas med följande ändring:

```
def half(x: Double): Double = x / 2.0
```

Lösn. uppg. 9. *Generiska klasser.*

a) –

b)

```
class Cell[T](var value: T):
  override def toString = "Cell(" + value + ")"
  def concat[U](that: Cell[U]): Cell[String] =
    Cell(s"$value${that.value}")
```

c) Endast celler med samma typparameter kan nu konkateneras. Eftersom `concat()` returnerar ett objekt av typ `Cell[String]` kan ett ojämnt antal celler med någon annan typparameter än `String` alltså inte längre konkateneras. Är antalet jämnt går det att konkatenera dem parvis och sedan konkatenera de returnerade `Cell[String]`-objekten, men det är något omständigt.

Lösn. uppg. 10. *Implementera fler generiska metoder i `Matrix[T]`.*

```
case class Matrix[T](data: Vector[Vector[T]]):
  require(data.forall(row => row.size == data(0).size))

  val dim: (Int, Int) = (data.length, data(0).length)

  def apply(row: Int, col: Int): T = data(row)(col)

  def updated(row: Int, col: Int)(value: T): Matrix[T] =
    Matrix(data.updated(row, data(row).updated(col, value)))

  def foreach(f: T => Unit): Unit = data.foreach(_.foreach(f))

  def foreachIndex(f: (Int, Int) => Unit): Unit =
    for r <- data.indices; c <- data(r).indices do f(r, c)

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

  def mapIndex[U](f: (Int, Int) => U): Matrix[U] =
    var result = Matrix.fill(dim)(f(0,0))
    for
      r <- data.indices
      c <- data(r).indices
    do
      result = result.updated(r, c)(f(r, c))
    end for
    result

  override def toString =
    s"""Matrix of dim $dim:\n${ data.map(_.mkString(" ")).mkString("\n") }"""

object Matrix:
  def fill[T](dim: (Int, Int))(value: T): Matrix[T] =
    Matrix[T](Vector.fill(dim._1, dim._2)(value))
```

9. Lösning `lookup`

9.1 Grunduppgifter; förberedelse inför laboration

Lösn. uppg. 1. *Para ihop begrepp med beskrivning.*

mängd	1	⇝	H	oordnad samling med unika element
nyckel-värde-tabell	2	⇝	F	oordnad samling av mappningar med unika nycklar
mappning	3	⇝	G	`nyckel -> värde`
nyckel	4	⇝	C	en unik identifierare
persistens	5	⇝	D	egenskapen att finnas kvar efter programmets avslut
serialisera	6	⇝	E	koda objekt till avkodningsbar sekvens av symboler
de-serialisera	7	⇝	B	avkoda symbolsekvens och återskapa objekt i minnet
linjärsöka	8	⇝	A	leta i sekvens tills sökkriteriet är uppfyllt

Lösn. uppg. 2. *Vad är en mängd?* En mängd är en samling som snabbt kan ge svaret på frågan om ett visst element ingår i samlingen eller ej. Elementen i en mängd är unika. Tilläg av redan existerande element ignoreras. En mängd är inte en sekvens, eftersom traversering med t.ex. `map` eller `foreach` inte (nödvändigtvis) sker i den ordning som elementen gavs när mängden konstruerades eller uppdaterades.

Lösn. uppg. 3. *Använda mängder.*

`Set(1, 2) ++ Set(1, 2)`	1	⇝	I	`Set(1, 2)`
`(1 to 3).toSet`	2	⇝	G	`Set(1) + 2 + 3`
`Vector.fill(3)(1).toSet`	3	⇝	F	`Set(1, 2) - 2`
`Set(1, 2, 3) diff Set(1, 2)`	4	⇝	B	`Set(3)`
`(1 to 7).toSet.apply(8)`	5	⇝	H	`false`
`Set(1, 2, 3).sorted`	6	⇝	D	`error: ...`
`Set(2,4) subsetOf (1 to 7).toSet`	7	⇝	E	`true`
`Set(1, -1, 2, -2).map(_.abs).sum`	8	⇝	A	`3`
`Set(1, 1, 1, 1, 1, 5).sum`	9	⇝	C	`6`

Lösn. uppg. 4. *Räkna unika ord med hjälp av en mängd.*

a)

```
scala> val hej = "hej hej hemskt mycket hej"
scala> val n = hej.split(' ').toSet.size
n: Int = 3
```

b) Metoden `distinct` returnerar en sekvens med unika element och bibehållen ursprunglig ordning.

Lösn. uppg. 5. *Skapa 2-tupler med metoden `->` som kan uttalas ”mappas till”.*

a) Ja, fabriksmetoden returnerar ett helt vanligt par:

```
scala> val härBorJag = "Skåne" -> "Lund"
val härBorJag: (String, String) = (Skåne,Lund)

scala> härBorJag._1
val res0: String = Skåne

scala> härBorJag._2
val res1: String = Lund
```

b)

```
val huvudstad = Vector(
  "Sverige"  -> "Stockholm",
  "Danmark"  -> "Köpenhamn",
  "Grönland" -> "Nuuk",
  "Skåne"    -> "Lund"
)
```

c)

```
scala> huvudstad(3)._2
val res2: String = Lund
```

Lösn. uppg. 6. *Linjärsöka efter nyckel i sekvens av mappningar.*

a)

```
def lookupIndex(xs: Vector[(String, String)])(key: String): Int =
  xs.indexWhere(_._1 == key)
```

b)

```
scala> val i = lookupIndex(huvudstad)("Skåne")
val i: Int = 3

scala> huvudstad(i)._2
val res2: String = Lund
```

Eller med funktioner som återanvändbara dellösningar:

```
scala> val indexOf = lookupIndex(huvudstad) _

scala> def capital(key: String) = huvudstad(indexOf(key))._2

scala> capital("Skåne")
val res3: String = Lund

scala> capital("Sverige")
val res4: String = Stockholm
```

Lösn. uppg. 7. *Nyckel-värde-tabell.*

```
scala> telnr("Fröken Ur")
val res0: Long = 464690510
```

```
scala> :type telnr
Map[String,Long]

scala> :type telnr.toVector
Vector[(String, Long)]
```

Lösn. uppg. 8. *Använda nyckel-värdetabell.*

a) Nej nyckel-värde-paren lagras i någon speciell ordning som bestäms av en intern, smart lagringsprincip enligt en s.k. hashfunktion[4], för att åstadkomma snabba uppslagningar av värden från nycklar och vilket normalt inte sammanfaller med ordningen i den sekvens som de skapades ur.

b)

`xs(2) + xs(4)`	1	⇝	A	`8`
`ys(0)`	2	⇝	C	`(10, 11)`
`xs(0)`	3	⇝	I	`NoSuchElementException`
`(xs + (0 -> 1)).apply(0)`	4	⇝	D	`1`
`xs.keySet.apply(2)`	5	⇝	G	**`true`**
`xs isDefinedAt 0`	6	⇝	H	**`false`**
`xs.getOrElse(0, 7)`	7	⇝	B	`7`
`xs.maxBy(_._2)`	8	⇝	E	`(16, 17)`
`xs.map(p => p._1 -> -p._2)(8)`	9	⇝	F	`-9`

Lösn. uppg. 9. *Registrering i förändringsbar nyckel-värde-tabell.*

```
class FreqMapBuilder:
  private val register = scala.collection.mutable.Map.empty[String,Int]
  def toMap: Map[String, Int] = register.toMap
  def add(s: String): Unit =
    register += (s -> (register.getOrElse(s, 0) + 1))

object FreqMapBuilder:
  def apply(xs: String*): FreqMapBuilder =
    val result = new FreqMapBuilder
    xs.foreach(result.add)
    result
```

Lösn. uppg. 10. *Metoden `sliding`.*

a)

```
scala> val xs = Vector("fem", "gurkor", "är", "fler", "än", "fyra", "tomater")
val xs: Vector[String] =
  Vector(fem, gurkor, är, fler, än, fyra, tomater)

scala> xs.sliding(2).toVector
```

[4] https://sv.wikipedia.org/wiki/Hashfunktion

```
val res9: Vector[Vector[String]] =
  Vector(Vector(fem, gurkor), Vector(gurkor, är), Vector(är, fler), Vector(fler

scala> xs.sliding(3).toVector
val res10: Vector[Vector[String]] =
  Vector(Vector(fem, gurkor, är), Vector(gurkor, är, fler), Vector(är, fler, är

scala> xs.sliding(10).toVector
val res11: Vector[Vector[String]] =
  Vector(Vector(fem, gurkor, är, fler, än, fyra, tomater))
```

`xs.sliding(n).toVector` skapar en sekvens som innehåller sekvenser av längden `n` som bildas genom att ta varje element och dess `n - 1` efterföljande element.

b)

```
scala> xs.sliding(2).map(ys => ys(0) -> ys(1)).toMap
val res0: Map[String,String] =
  Map(är -> fler,
      än -> fyra,
      fyra -> tomater,
      gurkor -> är,
      fem -> gurkor,
      fler -> än
  )
```

Man kan använda tabellen till att slå upp vilket som är efterföljande ord. Det fungerar eftersom alla ord är unika. Om det funnits flera likadana ord med olika efterföljande ord så hade vi behövt skapa en tabell med nycklar som mappar till en samling som registrerar efterföljande ord. Detta ska vi göra på veckans laboration.

Lösn. uppg. 11. *Läsa text från fil och webbservrar.*

a)

```
val populationOf = data.tail.map(v => v(0) -> v(1).toInt).toMap
val sizeOf       = data.tail.map(v => v(0) -> v(2).toInt).toMap
val capitalOf    = data.tail.map(v => v(0) -> v(3)).toMap
```

```
scala> capitalOf("Sverige")
res2: String = Stockholm

scala> populationOf("Sverige")
res3: Int = 9223766

scala> sizeOf("Sverige")
res4: Int = 449964
```

```
scala> val filename = "europa.txt"
scala> val xs = io.Source.fromFile(filename, "UTF-8").getLines.toVector
scala> val data = xs.map(_.split(';').toVector)
scala> data.map(_.map(_.take(15).padTo(15,' ')).mkString(" ")).foreach(println)
```

9.2 Extrauppgifter; träna mer

Lösn. uppg. 12. –

9.3 Fördjupningsuppgifter; utmaningar

Lösn. uppg. 13. *Registrering med `groupBy`.*

a) Metoden `groupBy` skapar en nyckel-värde-tabell där värdena i tabellen är en sekvens med elementen grupperade på ett speciellt sett. Mer precist:

Resultatet av `xs.groupBy(f: K => V)` för en sekvens `xs` av typen `Vector[K]` blir en tabell av typen `Map[V,Vector[K]]` där varje element `e` i `xs` är grupperade i samma tabellvärde om de lika är enligt `f(e)`. Varje grupp får tabellnyckeln `f(e)`.

Listigt trick: Om man låter funktionen `f` vara enhetsfunktionen som avbildar varje element på sig själv, alltså `x => x`, så grupperas värdena i samma sekvens om de är lika.

```
scala> val xs = Vector(1, 1, 2, 2, 4, 4, 4).groupBy(x => x > 2)
val xs: Map[Boolean,Vector[Int]] =
  Map(false -> Vector(1, 1, 2, 2), true -> Vector(4, 4, 4))

scala> val ys = Vector(1, 1, 2, 2, 4, 4, 4).groupBy(x => x)
val ys: Map[Int,Vector[Int]] =
  Map(2 -> Vector(2, 2), 4 -> Vector(4, 4, 4), 1 -> Vector(1, 1))
```

b)

```
def freq(xs: Vector[Int]): Map[Int, Int] =
  xs.groupBy(x => x).map(p => p._1 -> p._2.size)
```

Förklaring: metoden `groupBy` skapar en tabell med par `k, v` där `v` är en sekvens med så många `k` som antalet gånger `k` förekommer i `xs`. Genom att omvandla alla värden `p._2` till storleken `p._2.size` får vi en frekvenstabell.

```
scala> freq(kasta(1000))
val res0: Map[Int,Int] =
  Map(5 -> 163, 1 -> 174, 6 -> 161, 2 -> 169, 3 -> 167, 4 -> 166)

scala> freq(kasta(1000)).toVector.sortBy(_._1).foreach(println)
(1,183)
(2,167)
(3,169)
(4,179)
(5,154)
(6,148)
```

Lösn. uppg. 14. –

10. Lösning `inheritance`

10.1 Grunduppgifter; förberedelse inför laboration

Lösn. uppg. 1. *Para ihop begrepp med beskrivning.*

bastyp	1	⇝	N	den mest generella typen i en arvshierarki
supertyp	2	⇝	O	en typ som är mer generell
subtyp	3	⇝	D	en typ som är mer specifik
körtidstyp	4	⇝	J	kan vara mer specifik än den statiska typen
dynamisk bindning	5	⇝	L	körtidstypen avgör vilken metod som körs
polymorfism	6	⇝	B	kan ha många former, t.ex. en av flera subtyper
trait	7	⇝	I	är abstrakt, kan mixas in, kan ha parametrar
inmixning	8	⇝	H	tillföra egenskaper med `with` och en trait
överskuggad medlem	9	⇝	M	medlem i subtyp ersätter medlem i supertyp
anonym klass	10	⇝	C	klass utan namn, utvidgad med extra implementation
skyddad medlem	11	⇝	K	är endast synlig i subtyper
abstrakt medlem	12	⇝	F	saknar implementation
abstrakt klass	13	⇝	E	kan ha parametrar, kan ej instansieras, kan ej mixas in
förseglad typ	14	⇝	P	subtypning utanför denna kodfil är förhindrad
referenstyp	15	⇝	A	har supertypen `AnyRef`, allokeras i heapen via referens
värdetyp	16	⇝	G	har supertypen `AnyVal`, lagras direkt på stacken

Lösn. uppg. 2. *Gemensam bastyp.*

a) `Vector[Object]`. Typen `Object` i JVM är motsvarar typen `AnyRef` som är bastyp för alla referenstyper.

b) Felmeddelande:

```
scala> grönsaker.map(_.vikt).sum
-- Error:
1 |grönsaker.map(_.vikt).sum
  |              ^^^^^^
  |             value vikt is not a member of Object - did you mean wait?
-- Error:
1 |grönsaker.map(_.vikt).sum
  |                       ^
  |ambiguous implicit arguments: both object DoubleIsFractional in object Numer
```

Det första felmeddelandet beror på att vektorns element är av typen `Object` och medlemmen `vikt` är inte definierat för denna typ. Det andra felmeddelandet är ett följdfel som beror på att en sekvens med element av typen `Object` inte kan summeras eftersom kompilatorn inte kan härleda att elementtypen är numerisk.

c) Attributet `vikt` initialiseras vid konstruktion av `Gurka` resp. `Tomat`. Värdet ges av resp. klassparameter.

d) `Vector[Grönsak]`.

e) Ja. Eftersom den statiska typen för elementen i sekvensen är `Grönsak` (den dynamiska typen kan vara godtycklig subtyp av `Grönsak`) och alla instanser av denna

typ garanterat har attributet `vikt` som är av typen `Int` så kan kompilatorn vid *kompileringstid* dra slutsatsen att summeringen är giltig och därmed kan kompilatorn kompilera koden till körbar maskinkod.

f)

```
scala> new Grönsak
-- Error:
1 |new Grönsak
  |    ^^^^^^^
  |    Grönsak is a trait; it cannot be instantiated
```

g)

```
scala> val anonymGrönsak = new Grönsak { val vikt = 42 }
val anonymGrönsak: Grönsak = anon$1@1edde8b6
scala> anonymGrönsak.toString
val res0: String = anon$1@1edde8b6
```

Typen är `Grönsak` och blir här en s.k. *anonym klass*, eftersom vi inte har använt en namngiven klass med **`extends`**, utan bara ”hängt på” en klasskropp inom klammerparenteser direkt vid konstruktion. När du skapar anonyma klasser måste du använda nyckelordet **`new`**.

Kompilatorn hittar på ett unikt klassnamn, här anon$1, för att hålla reda på den anonyma klassen under kompilering till maskinkod. Strängrepresentationen innehåller ett hexadecimalt heltal som är unikt för instansen, här `1edde8b6`.

h)

```
scala> new Grönsak { }
-- Error:
1 |new Grönsak { }
  |^
  |object creation impossible, since val vikt: Int in trait Grönsak is not defined
```

Lösn. uppg. 3. *Polymorfism vid arv, s.k. subtypspolymorfism.*

a)

```
def skapaDjur(): Djur =
  if math.random() > 0.5 then Ko() else Gris()
```

b)

```
class Häst extends Djur:
  def väsnas = println("Gnäääääägg")

def skapaDjur(): Djur =
   math.random() match
    case r if r < 0.33 => Ko()
    case r if r < 0.67 => Gris()
    case _             => Häst()
```

Lösn. uppg. 4. *Olika typer av heltalspar till laborationen `snake0`.*

a)

```
trait Pair[T]:
  def x: T
  def y: T
  def tuple: (T, T) = (x, y)
```

b)

```
case class Dim(x: Int, y: Int) extends Pair[Int]
object Dim:
  def apply(dim: (Int, Int)): Dim = Dim(dim._1, dim._2)
```

c)

```
case class Pos private (x: Int, y: Int, dim: Dim) extends Pair[Int]:
  def +(p: Pair[Int]): Pos = Pos(x + p.x, y + p.y, dim)
  def -(p: Pair[Int]): Pos = Pos(x - p.x, y - p.y, dim)

object Pos:
  def apply(x: Int, y: Int, dim: Dim): Pos =
    import java.lang.Math.floorMod as mod
    new Pos(mod(x, dim.x), mod(y, dim.y), dim) //OBS: new nödvändig här!

  def random(dim: Dim): Pos =
    import scala.util.Random.nextInt as rni
    Pos(rni(dim.x), rni(dim.y), dim)
```

d) Om du glömmer skriva `new` explicit i kompanjonsobjektets `apply`-metod så blir det ett rekursivt anrop som resulterar i en oändlig loop vid körtid. Med `new` så är det garanterat den privata primärkonstruktorn för `Pos` som anropas.

I `Dim.apply` så skiljer sig parametertyperna åt mellan fabriksmetoden och primärkonstruktorn och kompilatorn väljer då primärkonstruktorn eftersom den passar med de givna två separata heltalen och inte med en 2-tupel.

e)

```
enum Dir(val x: Int, val y: Int) extends Pair[Int]:
  case North extends Dir( 0, -1)
  case South extends Dir( 0,  1)
  case East  extends Dir( 1,  0)
  case West  extends Dir(-1,  0)
export Dir.*  // gör så att North etc blir synliga i paketet snake
```

Lösn. uppg. 5. *Supertyp med parameter.*

a)

```
val person = new Person("Person1")
val akademiker = new Akademiker("Person2", "LTH")
val student = new Student("Person3", "LTH", "D")
val forskare = new Forskare("Person4", "LTH", "Doktorand")
```

b)

```
val vec = Vector(person, akademiker, student, forskare)
for(i <- vec){ print(i.toString + i.namn) }
```

c) Felmeddelande vid instansiering av `abstract class Akademiker`:
`Akademiker is abstract; it cannot be instantiated`

Det går *inte* lika bra med en `trait` i det speciella fallet `Akademiker`, eftersom en trait inte får skicka vidare parametrar till en supertyp. Felmeddelande:
`trait Akademiker may not call constructor of trait Person`

```
trait Person(val namn: String)

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

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

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

d)

```
scala>
     |trait Person(val namn: String)
     |
     | abstract class Akademiker(
     |   namn: String,
     |   val universitet: String) extends Person(namn)
     |
     | case class Student(
     |   namn: String,
     |   universitet: String,
     |   program: String) extends Akademiker(namn, universitet)
     |
     | case class Forskare(
     |   namn: String,
     |   universitet: String,
     |   titel: String) extends Akademiker(namn, universitet)
-- Error:
8 |  namn: String,
  |  ^
  |  error overriding value namn in trait Person of type String;
  |    value namn of type String needs `override` modifier
-- Error:
9 |  universitet: String,
  |  ^
  |  error overriding value universitet in class Akademiker of type String;
  |    value universitet of type String needs `override` modifier
-- Error:
13 |  namn: String,
   |  ^
   |  error overriding value namn in trait Person of type String;
   |    value namn of type String needs `override` modifier
```

```
-- Error:
14 |  universitet: String,
   |  ^
   |  error overriding value universitet in class Akademiker of type String;
   |    value universitet of type String needs `override` modifier
```

```
trait Person(val namn: String)

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

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

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

```
scala> val ps = Vector(Student("Kim", "Lund", "D"), Forskare("Herz", "Lund", "Dr"))
val ps: Vector[Akademiker] = Vector(Student(Kim,Lund,D), Forskare(Herz,Lund,Dr))
scala> ps :+ new Person("Abstrakt") {}
val res0: Vector[Person] =
  Vector(Student(Kim,Lund,D), Forskare(Herz,Lund,Dr), anon1@1941bbf3)
```

e)

```
trait Person:
  val namn: String
  val nbr: Long

trait Akademiker extends Person:
  val universitet: String

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

case class Forskare(
  namn: String,
  nbr: Long,
  universitet: String,
  titel: String) extends Akademiker

case class IckeAkademiker(
    namn: String,
```

```
    nbr: Long) extends Person
```

10.2 Extrauppgifter; träna mer

Lösn. uppg. 6. *Bastypen* `Shape` *och subtyperna* `Rectangle` *och* `Circle`.

a)

```
val c1 = Circle(Point(1, 1), 42)
val r1 = Rectangle(Point(3, 3), 20, 30)
c1.move(2, 3)
r1.move(3, 2)
```

b)

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

trait Shape:
  def pos: Point
  def move(dx: Double, dy: Double): Shape
  def moveTo(x: Double, y: Double): Shape

case class Rectangle(pos: Point, width: Double, height: Double) extends Shape:
  def move(dx: Double, dy: Double): Shape = copy(pos = pos.move(dx, dy))
  def moveTo(x: Double, y: Double): Shape = copy(pos.moveTo(x, y))

case class Circle(pos: Point, radius: Double) extends Shape:
  def move(dx: Double, dy: Double): Shape = copy(pos = pos.move(dx, dy))
  def moveTo(x: Double, y: Double): Shape = copy(pos.moveTo(x, y))
```

c) `def distanceTo(that: Point): Double = math.hypot(that.x - x, that.y - y)`

d) `def distanceTo(that: Shape): Double = pos.distanceTo(that.pos).`

e)

```
case class Point(x: Double, y: Double):
  def move(dx: Double, dy: Double): Point = Point(x + dx, y + dy)
  def moveTo(x: Double, y: Double): Point = Point(x, y)
  infix def distanceTo(that: Point): Double = math.hypot(that.x - x, that.y - y)

trait Shape:
  def pos: Point
  def move(dx: Double, dy: Double): Shape
  def moveTo(x: Double, y: Double): Shape
  infix def distanceTo(that: Shape): Double = pos.distanceTo(that.pos)

case class Rectangle(pos: Point, width: Double, height: Double) extends Shape:
  def move(dx: Double, dy: Double): Shape = copy(pos = pos.move(dx, dy))
  def moveTo(x: Double, y: Double): Shape = copy(pos.moveTo(x, y))

case class Circle(pos: Point, radius: Double) extends Shape:
  def move(dx: Double, dy: Double): Shape = copy(pos = pos.move(dx, dy))
  def moveTo(x: Double, y: Double): Shape = copy(pos.moveTo(x, y))
```

10.3 Fördjupningsuppgifter; utmaningar

Lösn. uppg. 7. *Inmixning.*

a) Det finns många olika sätt, några exempellösningar:

```
val antalFlygånkor: Int =
  fyle.count(f => f.isInstanceOf[Ånka] && f.ärFlygkunnig)
```

```
val antalFlygånkor: Int =
  fyle.filter(f => f.isInstanceOf[Ånka] && f.ärFlygkunnig).size
```

```
val antalFlygånkor: Int =
  fyle.collect{case f: Ånka if f.ärFlygkunnig}.size
```

```
val antalFlygånkor: Int = fyle.map(_ match
  case f: Ånka if f.ärFlygkunnig => 1
  case _ => 0
).sum
```

b)

```
val antalKrax: Int = fyle.filter(f => !f.ärSimkunnig).size * 2
val antalKvack: Int = fyle.filter(f => f.ärSimkunnig).size * 4
```

Lösn. uppg. 8. *Finala klasser.*

a) Sätt `final` framför `class` i klasserna.

b) error: illegal inheritance from final class Kråga.

Lösn. uppg. 9. *Accessregler vid arv och nyckelordet* ***`protected`****.*

a)

```
2 |  def avslöja = minHemlis
  |                ^^^^^^^^
  |                Not found: minHemlis
```

b)

```
scala> class Sub extends Super:
         def kryptisk = vårHemlis * math.Pi
scala> (new Sub).vårHemlis
-- Error:
1 |(new Sub).vårHemlis
  |^^^^^^^^^^^^^^^^^^
  |value vårHemlis in trait Super cannot be accessed as a member of Sub.
  | Access to protected value vårHemlis not permitted because enclosing object
  | is not a subclass of trait Super where target is defined
```

c) Ja.

Lösn. uppg. 10. *Användning av* ***protected****.*

a) I Fyle:

```
protected var räknaLäte: Int = 0
def väsnas: Unit = { print(läte * 2); räknaLäte += 2 }
```

I Ånka: `override def väsnas = { print(läte * 4); räknaLäte += 4 }`

b) `def antalLäten: Int = räknaLäte`

c) Om en klass som representerar en fågel som skulle ge ifrån sig fler/färre läten än en vanlig `Fyle`, behöver `väsnas` ändras. Denna metod behöver tillgång till `räknaLäte`, vilken inte får vara **`private`**.

d) Räknar-variabeln ska inte kunna påverkas i någon annan del av programmet.

Lösn. uppg. 11. *Inmixning av egenskaper.*

```
trait Fyle:
  val läte: String
  def väsnas: Unit = { print(läte * 2); räknaLäte += 2 }
  protected var räknaLäte: Int = 0
  val ärSimkunnig: Boolean
  val ärFlygkunnig: Boolean
  val ärStor : Boolean
  def antalLäten: Int = räknaLäte

trait KanSimma { val ärSimkunnig = true }
trait KanInteSimma { val ärSimkunnig = false }
trait KanFlyga { val ärFlygkunnig = true }
trait KanKanskeFlyga { val ärFlygkunnig = math.random() < 0.8 }
trait KanKanskeSimma { val ärSimkunnig = math.random() < 0.2 }
trait ÄrStor { val ärStor = true }
trait ÄrLiten { val ärStor = false }

final class Kråga extends Fyle, KanFlyga, KanInteSimma, ÄrStor:
  val läte = "krax"

final class Ånka extends Fyle, KanSimma, KanKanskeFlyga, ÄrStor:
  val läte = "kvack"
  override def väsnas = { print(läte * 4); räknaLäte += 4 }

final class Pjodd extends Fyle, KanFlyga, KanKanskeSimma, ÄrLiten:
  val läte = "kvitter"
  override def väsnas = { print(läte * 8); räknaLäte += 8 }
```

I REPL:

```
1 val fyle = Vector.fill(42)(
2   if math.random() < 0.33 then Kråga()
3   else if math.random() < 0.5 then Ånka()
4   else Pjodd()
5 )
6 fyle.filter(f => f.isInstanceOf[Kråga]).size * 2
7 fyle.filter(f => f.isInstanceOf[Ånka]).size * 4
```

```
8 fyle.filter(f => f.isInstanceOf[Pjodd]).size * 8
```

11. Lösning `context`

11.1 Grunduppgifter; förberedelse inför laboration

Lösn. uppg. 1. *Kontextparameter.*

a)

```
scala> given Int = 0
val res0: Int = 83

scala> f(using 41)
val res1: Int = 42

scala> g(41)(using 42)
val res2: Int = 83
```

Om man glömmer **`using`** vid explicit kontextargument blir det kompileringsfel. Kompilatorn blir ”förvirrad” och tror att du försöker ge ett ”vanligt” argument till en (i detta fallet) icke-existerande ”vanlig” parameterlista.

```
scala> f(41)
-- [E050] Type Error: ----------------------------------------------
1 |f(41)
  |^
  |method f does not take more parameters
  |
  | longer explanation available when compiling with `-explain`
1 error found

scala> :setting -explain

scala> f(41)
-- [E050] Type Error: ----------------------------------------------
1 |f(41)
  |^
  |method f does not take more parameters
  |----------------------------------------------------------------
    | Explanation (enabled by `-explain`)
  |- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  | You have specified more parameter lists than defined in the
    method definition(s).
   ----------------------------------------------------------------

scala> g(41)(42)
-- [E050] Type Error: ----------------------------------------------
1 |g(41)(42)
  |^^^^^
  |method g does not take more parameters
  |----------------------------------------------------------------
    | Explanation (enabled by `-explain`)
  |- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  | You have specified more parameter lists than defined in the
    method definition(s).
   ----------------------------------------------------------------
```

Det är inte vanligt att ange **`using`**-parametrar explicit; det vanligaste är att låta kompilatorn framkalla ett givet värde.

b) Det givna värdet `0` binds till motsvarande kontextparameter, som ska vara deklarerad i en egen parameterlista som börjar med **`using`**.

```
scala> f
val res3: Int = 1

scala> g(42)
val res4: Int = 42
```

c) Nej, det blir kompileringsfel om man försöker blanda vanliga parametrar och kontextparametrar i en och samma parameterlista:

```
scala> def h(i: Int, using j: Int) = i + j
-- [E040] Syntax Error: ----------------------------------------------
1 |def h(i: Int, using j: Int) = i + j
  |                    ^
  |                    ':' expected, but identifier found
```

Det är ett medvetet val att kräva separata parameterlistor, så att det inte ska uppstå förvirring om huruvida en vanlig parameter eller kontextparameter avses.

Lösn. uppg. 2. *Flera olika givna värden i lokal kontext.*

a) 2

b) 43

c) När kompilatorn försöker framkalla ett givet värde att automatiskt använda som argument till **`using`**-parametern `dx`, så letar den i den kontext som är närmast anropet först. Om det finns ett givet värdet i kompanjonsobjektet för parametertypen så tar kompilatorn detta i sista hand, om inget annat givet värde hittas närmare anropet.

Lösn. uppg. 3. *Lösning på konfigurationsproblemet med hjälp av givna värden.*

Nedan visas test av de tre olika lösningarna som givits i uppg. a)

Efter varje test diskuteras tillhörande för- och nackdelar, som efterfrågas i uppg. b)

```
def testGlobalVar(useDefault: Boolean = true) =
  import GlobalVar.*
  if useDefault then println(greetMsg) else
    GreetConfig.config = GreetConfig("Godmorgon", "världen")
    println(greetMsg)
```

Eftersom `config` här är en förändringsbar variabel, så kan en ändring på ett ställe påverka helt andra delar av programmet, vilket ibland kan vara en fördel, men ofta en nackdelen eftersom det kan vara svårt att förstå vad som händer bara genom att läsa en enskild del av programmet – en förändring av `config` kan ju ske varsomhelst. Det är lätt att glömma ändra till baka till default-värdet, om det är det som förväntas.

```
scala> testGlobalVar(); testGlobalVar(false); testGlobalVar()
Hello World!
Godmorgon världen!
Godmorgon världen!
```

```
def testDefaultArgs(useDefault: Boolean = true) =
  import DefaultArgs.*
```

```
3   if useDefault then println(greetMsg()) else
4     println(greetMsg(GreetConfig("Godmorgon","världen")))
```

Här sker ingen tillståndsförändring och default-användning är enkel, men det går inte enkelt att göra avsteg från default som gäller i en lokal kontext; vid *varje* enskilt anrop behöver du explicit ange alla de argument som inte ska vara default, så som visas på rad 4 ovan. Ändring av default har bara lokal påverkan. Om alla argument ska följa default, så gäller det att inte glömma anropa med tomt parentespar: `greetMsg()`. (Vad händer annars?)

```
1 scala> testDefaultArgs(); testDefaultArgs(false); testDefaultArgs()
2 Hello World!
3 Godmorgon världen!
4 Hello World!
```

Med kontextparametrar är flexibiliteten större; **`using`**-parametrar låter användaren själv styra vad som gäller i olika sammanhang och själva anropet blir enkelt oavsett om det är default-värdet eller andra, i den lokala kontexten, givna värden som önskas. Ändring av default har bara lokal påverkan, men den har påverkan på godtyckligt många anrop i den lokala kontexten – argument som skiljer sig kan alltså vara givna en gång utan att behöva upprepas vid varje anrop. Vid anrop där man vill låta kompilatorn framkallar givna värden för kontextparametern ska inga parenteser användas, och anropen bli därmed korta och enkla.

```
def testGivenVal(using g: GivenVal.GreetConfig) = println(g.greetMsg)
```

```
1  scala> testGivenVal
2  Hello World
3
4  scala> def localContext =
5           import GivenVal.*
6           given GreetConfig = GreetConfig("Godmorgon","världen")
7           testGivenVal
8
9  scala> localContext
10 Godmorgon världen
11
12 scala> testGivenVal
13 Hello World
```

c) Kompilatorn framkallar ett givet värde i den lokala kontexten:

```
1 scala> summon[GivenVal.GreetConfig]
2 val res0: GivenVal.GreetConfig = GreetConfig(Hello,World)
```

Kompilatorn följer denna prioritetsordning i sökandet efter ett unikt givet värde:

1. **Explicita** argument till kontextparametrar märkta med **`using`**
2. **`given`** och **`import given`** ... i aktuell namnrymd (eng. *current scope*)
3. **`given`**-värden i **kompanjonsobjekt** för den använda typen.

Om flera givna värden kan framkallas för typer som ingår i en gemensam arvshierarki så väljer kompilatorn det givna värdet som är av den *mest specifika* typen.

d) Det blir kompileringsfel om kompilatorn inte hittar ett givet värde för den typ som avses.

```
scala> summon[Long]
-- Error: ----------------------------------------------------------
1 |summon[Long]
  |            ^
  |            no given instance of type Long was found for parameter x of
  |            method summon in object Predef
1 error found
```

e) Ja! Det får *inte* vara tvetydigt vilket givet värde som ska framkallas:

```
scala> def tvetydigt =
     |   given a: Int = 42
     |   given b: Int = 43
     |   summon[Int]
     |
-- Error: ----------------------------------------------------------------
4 |  summon[Int]
  |            ^
  |ambiguous given instances: both given instance b and given instance a
  |match type Int of parameter x of method summon in object Predef
1 error found
```

Läs mer om kontexuella abstraktioner här:
https://docs.scala-lang.org/scala3/reference/contextual/

11.2 Extrauppgifter; träna mer

Lösn. uppg. 4. *Kontextparameter och givet värde.*

a)

```
scala> add(1)
-- Error: ----------------------------------------------------------------
1 |add(1)
  |     ^
  |    no given instance of type Int was found for parameter y of method add
1 error found
```

b) Nu finns ett givet värde som kompilatorn automatiskt kan fylla i på platsen vid anropet.

c) `def sub(x: Int)(using y: Int) = x - y`

11.3 Fördjupningsuppgifter; utmaningar

Lösn. uppg. 5. *Varians och typgränser.*

a) Gör lådan flexibel i sin typparameter med ett + före typparametern enligt nedan.

```
case class Box[+A](x: A)
```

Kompilatorn tillämpar reglerna för kovarians eftersom typparametern har ett plustecken framför sig: `Box[Cat]` är en suptyp till `Box[Any]` om `Cat` är en subtyp till `Any`, vilket den ju är eftersom alla typer är subtyp till `Any`.

b) Förklaringen till beteendet har med olika varians att göra:

- Samlingen `Vector` är kovariant och därmed flexibel i sin typparameter (liksom andra oföränderliga sekvenser i Scalas standardbibliotek). Kompilatorn betraktar därmed `Vector[Cat]` som en subtyp till `Vector[Pet]` eftersom `Cat` är en subtyp till `Pet`. På platser i koden där en `Vector[Pet]` krävs så anses `Vector[Cat]` överensstämma med (eng. *conforms to*) `Vector[Pet]` och får därmed duga på dessa platser.
- En mängd har en apply-metod från elemttypen till `Boolean` som ger innehållstest. Av det skälet har man låtit `Set[T]` ärva `Function1[T, Boolean]` som är deklarerad kontravariant i `T`, så att en mängd kan användas där en `T => Boolean` förväntas. Även om det skulle vara praktiskt om Set[T] vore kovariant i `T`, i likhet med `Vector`, `List`, `Seq` etc, så kan inte `T` vara både kovariant och kontravariant på en och samma gång. Man har därför valt att göra `Set` invariant och därmed är mängder ej flexibla i sin typparameter. `Set[Cat]` är alltså *inte* en subtyp till `Set[Pet]` *även* om `Cat` är en subtyp till `Pet`, vilket ger kompileringsfel i uppgiftens exempel. Se även https://stackoverflow.com/questions/676615/why-is-scalas-immutable-set-not-covariant-in-its-type
- Med `:settings -explain` ger kompilatorn en längre utskrift som förklarar den bevisföring som skedde under kompileringens typkontroll.

c) Det blir kompileringsfel då metoden `isHealthy` ej existerar för godtycklig typ.

d) Lägg till en övre gräns som garanterar att metoden `isHealthy` finns för alla typer som kan bindas till typparametern `A`:

```
class Vet[-A <: Pet]:
  def treat(x: A): Unit = x.isHealthy = true
```

Kompilatorn garanterar alltså att typparametern `A` är ”mindre än eller lika med” `Pet`.

e) Veterinären `Vet` är flexibel i sin typparameter och minustecknet anger kontravarians och därmed att `Vet[Pet]` är en subtyp till `Vet[Cat]` då `Cat` är en subtyp till `Pet`. Detta kan demonstreras med nedan exempel:

```
1  scala> val pinkPanther = Cat()
2  val pinkPanther: Cat = Cat@33e7ece5
3
4  scala> val somePet: Pet = Cat()
5  val somePet: Pet = Cat@57f1cb96
6
7  scala> val catVet = Vet[Cat]()
8  val catVet: Vet[Cat] = Vet@1060e784
9
10 scala> pinkPanther.isHealthy = false
11
12 scala> catVet.treat(pinkPanther)
13
14 scala> pinkPanther.isHealthy
15 val res2: Boolean = true
16
17 scala> somePet.isHealthy = false
18
19 scala> catVet.treat(somePet)
20 -- [E007] Type Mismatch Error: --------------
```

```
1 |catVet.treat(somePet)
  |             ^^^^^^^
  |             Found:    (somePet : Pet)
  |             Required: Cat

scala> val powerVet = Vet[Pet]()
val powerVet: Vet[Pet] = Vet@2eb90ae9

scala> pinkPanther.isHealthy = false

scala> powerVet.treat(pinkPanther)

scala> pinkPanther.isHealthy
val res3: Boolean = true

scala> val pluto = Dog()
val pluto: Dog = Dog@6f27db5d

scala> pluto.isHealthy = false

scala> powerVet.treat(pluto)

scala> pluto.isHealthy
val res4: Boolean = true
```

Lösn. uppg. 6. *Typklasser och kontextparametrar.*

a) TODO!!!

b) TODO!!!

c) TODO!!!

Lösn. uppg. 7. *Användning av given ordning.*— TODO!!!

Lösn. uppg. 8. *Skapa egen implicit ordning med `Ordering`.*

a) TODO!!!

b) TODO!!!

c) TODO!!!

d) TODO!!!

Lösn. uppg. 9. *Specialanpassad ordning genom att ärva från `Ordered`*

a)

```
case class  Team(name: String, rank: Int) extends Ordered[Team]:
  override def compare(that: Team): Int = -rank.compare(that.rank)
```

b)

```
scala> Team("fnatic",1499) < Team("gurka", 2)
val res1: Boolean = true
```

c) Ad hoc polymorfism är mer flexibel. TODO!!! mer diskussion om likheter och skillnader här...

12. Lösning extra

12.1 Uppgifter om sökning och sortering

Lösn. uppg. 1. *Tidmätning.*

a) Exekvera koden och du bör finna att det tar längre tid att hitta värdet 1 i vårt Set s än i vektorn v.

b)

En vektor har en sekventiell ordning som find kan använda, medan `Set` är internt ordnad på ett annat sätt för att innehållskontroll ska gå extra snabbt. Anledningen att det tar tid för `find` på `Set` är att det först måste skapas en iterator innan vår mängd kan gås igenom från början till slut. Metoden `contains` på `Set` däremot är rasande snabb beroende på den interna strukturen hos objekt av typen `Set` (som är smart designad med s.k. hash-koder, där det går lika snabbt att hitta ett element oavsett vart det befinner sig).

Lösn. uppg. 2. *Sökning med inbyggda sökmetoder.*

a) Förslag på test av `indexOfSlice`:

```
scala> List(1,2,3,35,1,23).indexOfSlice(List(35,1,23))
res73: Int = 3
scala> List(1,2,3,35,1,23).indexOfSlice(List(35,1,3))
res74: Int = -1
```

b) Förslag på test av `lastIndexOfSlice`:

```
Vector(1,2,3,4,1,2).lastIndexOfSlice(Vector(1,2))
res2: Int = 4
Vector("apa", "banan", "majs", "banan").lastIndexOfSlice(Vector("banan"))
res3: Int = 3
Vector("apa", "banan", "majs", "banan").lastIndexOfSlice(Vector("banand"))
res4: Int = -1
```

c) Observera att metoden `search` antar att samlingen är sorterad i stigande ordning. När vi inverterar ordningen kan `search` oftast inte hitta det vi letar efter, eftersom den kommer leta i fel halva av samlingen.

```
scala> val udda = (1 to 1000000 by 2).toVector
scala> import scala.collection.Searching._
scala> udda.search(udda.last)
res18: collection.Searching.SearchResult = Found(499999)
//Search hittar det sista elementet på plats 499999 i samlingen.

scala> udda.search(udda.last + 1)
res19: collection.Searching.SearchResult = InsertionPoint(500000)
//Search kan inte hitta udda.last + 1 eftersom det inte existerar i samlingen
//och returnerar således ett objekt av typen InsertionPoint med värdet 500000.
//Vårt element udda.last + 1 hade alltså legat på plats 500000 om det funnits.

scala> udda.reverse.search(udda(0))
res20: collection.Searching.SearchResult = InsertionPoint(0)
//Som förklarat innan så förutsätter search att listan är sorterad i stigande
//ordning, så den kan inte hitta elementet udda(0) = 1 när listan är inverterad
```

```
scala> def lin(x: Int, xs: Seq[Int]) = xs.indexOf(x)
scala> def bin(x: Int, xs: Seq[Int]) = xs.search(x) match
  case Found(i) => i
  case InsertionPoint(i) => -i

//Definierar en metod bin som använder sig av metoden search på en sekvens.
//Den ser sedan till med hjälp av "pattern matching" att bara returnera positio
//i, och inte ett objekt av typen Found eller InsertionPoint.

scala> timed{ lin(udda.last, udda) }
time: 42.294821 ms
res22: (Int, Long) = (499999,42294821)
//För att hitta udda.last = 499999 med linjärsökning tog det ca 42ms.

scala> timed{ bin(udda.last, udda) }
time: 0.147314 ms
res23: (Int, Long) = (499999,147314)
//Binärsökning för att hitta värdet 499999 tog extremt mycket kortare tid.
//Detta för att vid varje steg i binärsökningen halveras mängden tal som
//sökningen måste kolla i. Detta är dock ett extremfall eftersom vi söker
//talet längst bak i listan. Om vi istället gjort en linjärsökning efter
//det första talet 1, hade detta gått minst lika snabbt som binärsökning.
```

d) Det behövs $log_2(n)$ jämförelser. Detta eftersom att vi hela tiden halverar antalet element i listan vi behöver söka igenom. Så efter första jämförelsen har vi $\frac{n}{2}$ element kvar. Efter andra jämförelsen har vi $\frac{n}{2*2}$ element kvar etc. När vi bara har ett element kvar har vi hittat det vi söker efter, och har då gjort b antal jämförelser. Ekvationen ser då ut på följande vis:

$$\frac{n}{2^b} = 1$$

Enligt lagarna för logaritmer kan vi nu komma fram till vad b är:

$$log_2(n) = b$$

Lösn. uppg. 3. *Sök bland LTH:s kurser med linjärsökning.*

a) Första raden innehåller kolumnnamnen `Kurskod KursSve KursEng Hskpoang Niva`. Därefter kommer en rad för varje kurs med kursdata enligt kolumnnamnen.

b) Koden laddar ner data och skapar en vektor med instanser av case-klassen `Course` med hjälp av metoden `fromLine`. Eftersom variabeln `lth` är deklarerad som **`lazy`** kommer inte `download()` bli anropad förrän första gången som variablen `lth` refereras. Antalet kurser ges av:

```
scala> val n = courses.lth.length
n: Int = 1104
```

c)

```
1 scala> def isCS(s: String) = s.startsWith("EDA") || s.startsWith("ETS")
2 scala> val x = courses.lth.find(c => isCS(c.code) && c.level == "G2")
3 x: Option[courses.Course] = Some(Course(EDAF05,Algoritmer, datastrukturer och
4    komplexitet,Algorithms, Data Structures and Complexity,5.0,G2))
```

d)

```
def linearSearch[T](xs: Seq[T])(p: T => Boolean): Int =
   var i = 0
   while(i < xs.length && !p(xs(i))) i += 1
   if (i < xs.length) i else -1
```

e)

```
def rndCode: String =
   //randomizes from 0 to n (inclusive)
   def rnd(n: Int) = (math.random() * (n + 1)).toInt

   def letter = (rnd('Z' - 'A') + 'A').toChar
   def dig = ('0' + rnd(9)).toChar
   Seq(letter, letter, letter, letter, dig, dig).mkString
```

f)

```
val xs = Vector.fill(500000)(rndCode)
val(ixs, elapsedLin) =
  timed { xs.map(x => linearSearch(courses.lth)(_.code == x)) }
val found = ixs.filterNot(_== -1).size
```

g)

```
def linearSearch[T](xs: Seq[T])(p: T => Boolean): Int = xs.indexWhere(p)
```

Lösn. uppg. 4. *Sök bland LTH:s kurser med binärsökning.*

a) —

b)

```
def binarySearch(xs: Seq[String], key: String): Int =
  var (low, high) = (0, xs.size - 1)
  var found = false
  var mid = -1

  while (low <= high && !found) do
    mid = (low + high) / 2
    if xs(mid) == key then found = true
    else if xs(mid) < key then low = mid + 1
    else high = mid - 1
  end while
  if found then  mid else -(low + 1)
```

c) Med en i7-3770K @ 3.50Hz tog sökningarna följande tid:

- Binärsökning: `time: 142.6 ms`
- Linjärsökning: `time: 3316.5 ms`

Med en i7-8700T @ 2.40GHz tog sökningarna följande tid:

- Binärsökning: `time: 81.5 ms`
- Linjärsökning: `time: 5138.6 ms`

d) Binärsökningen var ca 23 gånger snabbare på en i7-3770K @ 3.50Hz och ca 63 gånger snabbare på en i7-8700T CPU @ 2.40GHz.

Lösn. uppg. 5. *Insättningssortering.*

a) —

b)

```
def insertionSort(xs: Seq[Int]): Seq[Int] =
  val result = scala.collection.mutable.ArrayBuffer.empty[Int]
  for e <- xs do
    var pos = 0
    while pos < result.size && result(pos) < e do pos += 1
    result.insert(pos,e)
  end for
  result.toVector
```

Lösn. uppg. 6. *Sortering på plats.*

```
def selectionSortInPlace(xs: Array[String]): Unit =
  def indexOfMin(startFrom: Int): Int =
    var minPos = startFrom
    var i = startFrom + 1
    while (i < xs.size) do
      if (xs(i) < xs(minPos)) minPos = i
      i += 1
    end while
    minPos
  end indexOfMin

  def swapIndex(i1: Int, i2: Int): Unit =
    val temp = xs(i1)
    xs(i1) = xs(i2)
    xs(i2) = temp
  end swapIndex

  for i <- 0 to xs.size - 1 do swapIndex(i, indexOfMin(i))
end selectionSortInPlace
```

Lösn. uppg. 7. *Undersök om en sekvens är sorterad.*

a) Det tar i värsta fall $O(n * log(n))$ för timsort att sortera listan med n element. Sedan krävs n stycken jämförelser mellan den sorterade och osorterade listan. Det totala antalet jämförelser i värstafallet uppgår därför till max $n + n * log(n)$. För 10^6 element blir det ca 10^7 jämförelser.

```
scala> val n = 1E6
val n: Double = 1000000.0

scala> def worstCase(n: Double) = n + n * math.log(n)
def worstCase(n: Double): Double

scala> println(s"i värsta fall med n=$n så blir det ${worstCase(n)} jämförelse
i värsta fall med n=1000000.0 så blir det 1.4815510557964273E7 jämförelser
```

b) En mer effektiv version av `isSorted` som avbryter sökningen när ett osorterat element upptäcks:

```
def isSorted(xs: Vector[Int]): Boolean =
  if xs.length > 1 then
    var i = 0
    var result = true
    while i < xs.length-1 && result do
      if xs(i) > xs(i+1) then result = false
      i += 1
    end while
    result
  else true
end isSorted
```

c) I värsta fall behöver man göra $n - 1$ parvisa jämförelser, om alla ligger i sorterad ordning utom den sista.

d) 2-tupeln är av typen `(Int, Int)`.

```
def isSorted(xs: Vector[Int]): Boolean =
  xs.zip(xs.tail).forall(x => x._1 <= x._2)
```

Lösn. uppg. 8. *Insättningssortering på plats.*

```
def insertionSort(xs: Array[Int]): Unit =
  for elem <- 1 until xs.length if xs.length > 0 do
    var pos = elem
    while pos > 0 && xs(pos) < xs(pos - 1) do
      val temp = xs(pos -1)
      xs(pos -1) = xs(pos)
      xs(pos) = temp
      pos -= 1
    end while
  end for
end insertionSort
```

Lösn. uppg. 9. *Sortering till ny sekvens med urvalssortering.*

```
def selectionSort(xs: Seq[String]): Seq[String] =
  def indexOfMin(xs: Seq[String]): Int = xs.indexOf(xs.min)
  val unsorted = xs.toBuffer
  val result = scala.collection.mutable.ArrayBuffer.empty[String]
  while !unsorted.isEmpty do
    val minPos = indexOfMin(unsorted)
    val elem = unsorted.remove(minPos)
    result.append(elem)
  end while
  result.toVector
end selectionSort
```

12.2 Uppgifter om trådar och jämlöpande exekvering

Lösn. uppg. 10. *Trådar.*

a) -

b) `java.lang.IllegalThreadStateException`. Det går inte att starta en tråd mer än en gång. Tråden kan därför inte startas om när den redan har exekverats.

c) När `start` anropas exekveras koden i `run` parallellt. Därför skrivs `Gurka` och `Tomat` ut omlöpande. Om istället `run` anropas direkt blir det inte jämnlöpande exekvering och `Gurka` skrivs ut 100 gånger, sedan skrivs `Tomat` ut 100 gånger.

d) `Thread.sleep` pausar inte tråden i exakt den tiden som angets. Alltså kommer det skrivas ut `zzz snark hej!` i de flesta fall, men det är inte garanterat.

Lösn. uppg. 11. *Jämlöpande variabeluppdatering.*

a) I `slösaSpara` hämtas saldot, ändras och placeras tillbaka i minnet - fördröjs - upprepas. Om `bamse` blir klar med att ladda, ändra och lagra innan skutt gör detsamma blir det problem, då de tävlar om vem som får uppdatera (eng. *race contiion*). Problemet innan en tråd kan lagra det förändrade värdet laddar den andra tråden det gamla värdet. Bara en av dessa trådar vinner racet och får lagra sitt ändrade tal och den andra ändringen går förlorad. `skutt` och `bamse` blir alltså upprörda för att inte alla dess uttag och insättningar registreras.

Lösn. uppg. 12. *Trådsäkra `AtomicInteger`.*

Nu är `farmor`-tråden garanterad att kunna ladda saldot, ta ut pengar/ändra och lagra innan `vargen`-tråden kan skriva över resultatet. I `slösaSpara` pausas tråden i en millisekund så `vargen`-tråden kan hinna ta ut pengar innan `farmor`-sätter hinner sätta in pengar igen och saldot blir negativt. Dock kommer alla uttag och insättningar registreras eftersom operationerna är atomära och saldot kommer återställas till noll, utan att insättningar går förlorade.

Lösn. uppg. 13. *Jämlöpande exekvering med `scala.concurrent.Future`.*

a) error: Cannot find an implicit ExecutionContext. Future behöver en ExecutionContext för att kunna köras. `f` är av typen Future[Unit].

b) Funktionen `printLater` har en Future, vilket innebär att när både `printLater` och `println` anropas i foreach-loopen exekveras de jämnlöpande. Eftersom det tar

längre tid att starta upp en Future för datorn är `println` snabbare och skriver ut att alla är igång först. Sedan skrivs siffrorna från 1 - 42 ut med oregelbundna mellanrum eftersom tråden pausas olika länge.

c) `big` är en Future[Int]. Det stora talet har 7 520 383 siffror. `r` är av typen Try[Int] (se dokumentationen för Future om du är osäker)

d) Eftersom exekveringen blockas tills den har fått ett resultat går det inte att fortsätta skriva i REPL medan uträkningen pågår. Väntar man för kort tid får man ett TimeOutException och uträkningen avbryts.

Lösn. uppg. 14. *Använda `Future` för att göra flera saker samtidigt.*

a) -

b) -

c) Varje sida fördröjs med mellan 2 upp till 3 sekunder (2000-3000 millisekunder). Så i medeltal tar det 2.5 sekunder för varje sida att laddas. Vektorn måste fyllas innan exekveringen kan fortsätta. Därför laddas alla 10 stycken sidor in innan man kan se första websidan. Det tar därför i medeltal 2.5 x 10 = 25 sekunder.

d) `f` ger en Vektor fylld med strängar där varje element ges av en rad på hemsidan. Då `f` körs i bakgrunden kan programmet fortlöpa medan innehållet räknas ut. Du kan därför skriva `f` i REPL:n men det är inte säkert att processen är klar och det slutgiltiga resultatet visas.

e) Samma som ovan, förutom att det blir en vektor där varje element är i sig en vektor med strängar.

f) Ladda data parallellt så att nedladdningen sker samtidigt, men det går olika snabbt pga metoden seg.

g) Eftersom datan laddas i parallella trådar utan blockering blir de inte klara i ordning, utan i den ordningen tråden körs klart. Till slut blir alla klara och resultatet visar en vektor med **`true`** värden.

h) Metoden `lycka` är väldefinerad och kastar därför inga undantag. Den skriver alltid ut `:)`. Metoden `olycka` är inte väldefinierad då division med 0 ger `java.lang.ArithmeticException`. Detta fångas upp vid callbacken och det skrivs ut `:(` samt det specificerade undantaget.

12.3 Extrauppgifter; träna mer

Lösn. uppg. 15.

```
def isPrime(n: BigInt): Boolean = n match
  case _ if (n <= 1) => false
  case _ if (n <= 3) => true
  case _ if n % 2 == 0 || n % 3 == 0 => false
  case _ =>
    var i = BigInt(5)
    while i * i < n do
      if (n % i == 0 || n % (i + 2) == 0) false
      i += 6
    end while
    true
end isPrime
```

```
import scala.concurrent.*
import ExecutionContext.Implicits.global

val primes = Vector.fill(10)(Future{nextPrime(randomBigInt(16))})
primes.foreach(_.onSuccess{case i => println(i)})
```

Lösn. uppg. 16. *Svara på teorifrågor.*

a) Stackoverflow ger följande förklaring:
A thread is an independent set of values for the processor registers (for a single core). Since this includes the Instruction Pointer (aka Program Counter), it controls what executes in what order. It also includes the Stack Pointer, which had better point to a unique area of memory for each thread or else they will interfere with each other.

b)

```
val thread = new Thread(new Runnable{
  def run(){println(''Det här är en tråd'')}
})
```

c) `thread.start`

d) Det kan bli kapplöpning(race conditions) om vilken tråds resurser blir sparade. Vilket leder till att de andra trådarnas ändringar blir ignorerade.

e) Trådsäkerhet innebär att flera trådar kan köras parallellt utan felaktigheter i resultatet. Exempelvis får man vara väldigt försiktig om man vill ha en muterbar variabel som alla trådar ska ändra samtidigt.

f) Till exempel slipper man skapa instanser av klassen Thread eftersom man kan placera koden i en Future istället. Den löser även mycket under huven för kodaren.

Lösn. uppg. 17. –

Lösn. uppg. 18. *Skapa din egen multitrådade webbserver.*

a) `abbasillen` skrivs ut baklänges till `nellisabba`.

b)

c)

d)

e)

f)

g)

h)

i)
Lösningsförslag:

```
1 package fibserver.threaded.memcache.whileloop
2
3 import java.net.{ServerSocket, Socket}
4 import java.io.OutputStream
5 import java.util.Scanner
6 import scala.util.{Try, Success, Failure}
7 import scala.concurrent._
```

```
8  import ExecutionContext.Implicits.global
9
10 object html:
11   def page(body: String): String =  //minimal web page
12     s"""<!DOCTYPE html>
13        |<html><head><meta charset="UTF-8"><title>Min Sörver</title></head>
14        |<body>
15        |$body
16        |</body>
17        |</html>
18        """.stripMargin
19
20   def header(length: Int): String = //standardized header of reply to client
21     s"HTTP/1.0 200 OK\nContent-length: $length\nContent-type: text/html\n\n"
22
23   def insertBreak(s: String, n: Int = 80): String =
24     if s.size < n then s else s.take(n) + "</br>" + insertBreak(s.drop(n),n)
25
26 object compute:
27   import java.util.concurrent.ConcurrentHashMap
28   val memcache = new ConcurrentHashMap[BigInt, BigInt]
29
30   def fib(n: BigInt): BigInt =
31     if memcache.containsKey(n) then
32       println("CACHE HIT!!! no need to compute: " + n)
33       memcache.get(n)
34     else
35       println("cache miss :( must compute fib:  " + n)
36       val f = superFib(n)
37       memcache.put(n, f)
38       f
39
40   private def superFib(n: BigInt): BigInt =
41     if n <= 0 then 0
42     else if n == 1 || n == 2 then 1
43     else
44       var secondLast: BigInt = 1
45       var last: BigInt = 1
46       var sum: BigInt = secondLast + last
47       var i = 3
48       while i < n do
49         if memcache.containsKey(i) then
50           sum = memcache.get(i)
51         else
52           secondLast = last
53           last = sum
54           sum = secondLast + last
55           memcache.put(i, sum)
56         i += 1
57       sum
58
59
60 object start:
61
62   def fibResponse(num: String) =
63     num.toIntOption match
64       case Some(n) => html.page(s"fib($n) == " + compute.fib(n))
```

```
65        case None    => html.page(s"FEL: skriv ett heltal, inte $num")
66
67    def errorResponse(uri:String) = html.page(s"Error: $uri </br> use /fib/heltal")
68
69    def handleRequest(cmd: String, uri: String, socket: Socket): Unit =
70      val os = socket.getOutputStream
71      val afterSlash = uri.toString.drop(1) // skip initial slash
72      println(s"afterSlash:$afterSlash")
73      val response: String =
74        if afterSlash.startsWith("fib/") then fibResponse(afterSlash.stripPrefix("fib/"))
75        else errorResponse(uri)
76      os.write(html.header(response.size).getBytes("UTF-8"))
77      os.write(response.getBytes("UTF-8"))
78      os.close
79      socket.close
80    end handleRequest
81
82    def serverLoop(server: ServerSocket): Unit =
83      println(s"http://localhost:${server.getLocalPort}/hej")
84      while true do
85        Try {
86          var socket = server.accept  // blocks thread until connect
87          val scan = new Scanner(socket.getInputStream, "UTF-8")
88          val (cmd, uri) = (scan.next, scan.next)
89          println(s"Request: $cmd $uri")
90          Future { handleRequest(cmd, uri, socket) }.recover {
91            case e => println(s"Reqest failed: $e")
92          }
93        }.recover{ case e: Throwable => s"Connection failed: $e" }
94
95    def main(args: Array[String]) =
96      val port = Try{ args(0).toInt }.getOrElse(8089)
97      serverLoop(new ServerSocket(port))
```

13. Lösning examprep

Lösn. uppg. 1. *Gör klart ditt projekt.* —

Lösn. uppg. 2. *Gör en extenta.* —

Lösn. uppg. 3. *Förbered din projektredovisning.* –

Lösn. uppg. 4. *Skapa dokumentation för ditt projekt.*–

Lösn. uppg. 5. *Repetera övningar och laborationer.* –