EDAA45 Programmering, grundkurs

Läsvecka 2: Kodstrukturer

Björn Regnell

Datavetenskap, LTH

Lp1-2, HT 2016

Studieteknik

Hur studerar du?

- Vad är bra **studieteknik**?
- Hur lär **du** dig bäst? Olika personer har olika preferenser.
  - Ta reda på vad som funkar bäst för dig.
  - En kombination av flera sinnen är bäst: läsa+prata+skriva...
  - Aktivera dig! Inte bara passivt läsa utan också aktivt göra.
- Hur skapa **struktur**? Du behöver ett sammanhang, ett **system av begrepp**, att **placera in** din nya kunskap i.
- Hur uppbåda **koncentration**? Steg 1: Stäng av mobilen!
- Hur vara **disciplinerad**? Studier först, nöje sen!
- Du måste **planera och omplanera** för att säkerställa **tillräckligt mycket egen pluggtid** då du är pigg och koncentrerad för att det ska funka!
- Programmering **kräver** en **pigg och koncentrerad hjärna**!

Hur ska du studera programmering?

- När du gör **övningarna**:
  - Ta fram föreläsningsbilderna i pdf och kolla igenom dem.
  - Är det något i föreläsningsbilderna du inte förstår: ta upp det i samarbetsgrupperna eller på resurstiderna.
  - Om något är knepigt:
    - Hitta på egna REPL-experiment och undersök hur det funkar.
    - Följ ev. länkar i föreläsningsbilderna, eller googla själv på wikipedia, stackoverflow, ...
- När du gör **laborationerna**:
  - Kolla igenom målen för veckans **övning** och dubbelkolla så att du har uppnått dem.
  - Gör förberedelserna **i god tid innan** labben.
  - Läs igen **hela** labbinstruktionen **innan** labben.

På rasten: träffa din samarbetsgrupp

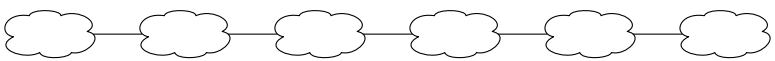
- Träffas i samarbetsgrupperna och bestäm/gör/diskutera:
  1. När ska ni träffas nästa gång?
  2. Bläddra igenom föreläsningsbilderna från w01 i pdf.
  3. Vilka **koncept** är fortfarande (mest) **grumliga**?
     Alltså: Vilka koncept från förra veckan vill ni på nästa möte jobba mer med i gruppen för att alla ska förstå grunderna?

Datastrukturer och kontrollstrukturer

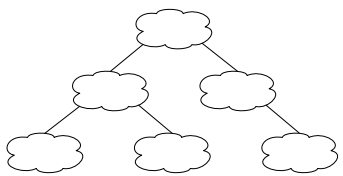
Vad är en datastruktur?

- En datastruktur är en struktur för organisering av data som...
  - kan innehålla **många** element,
  - kan refereras till med **ett** enda namn, och
  - ger möjlighet att komma åt de enskilda elementen.
- En **samling** (eng. *collection*) är en datastruktur som kan innehålla många element av **samma typ**.
- Exempel på olika samlingar där elementen är organiserade på olika vis:

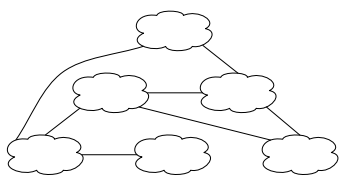
Lista



Träd



Graf



Mer om listor & träd fördjupningskursen. Mer om träd, grafer i Diskreta strukturer.

Vad är en vektor?

En **vektor**[1] (eng. *vector, array*) är en **samling** som är **snabb** att **indexera** i. Åtkomst av element sker med `apply(platsnummer)`:

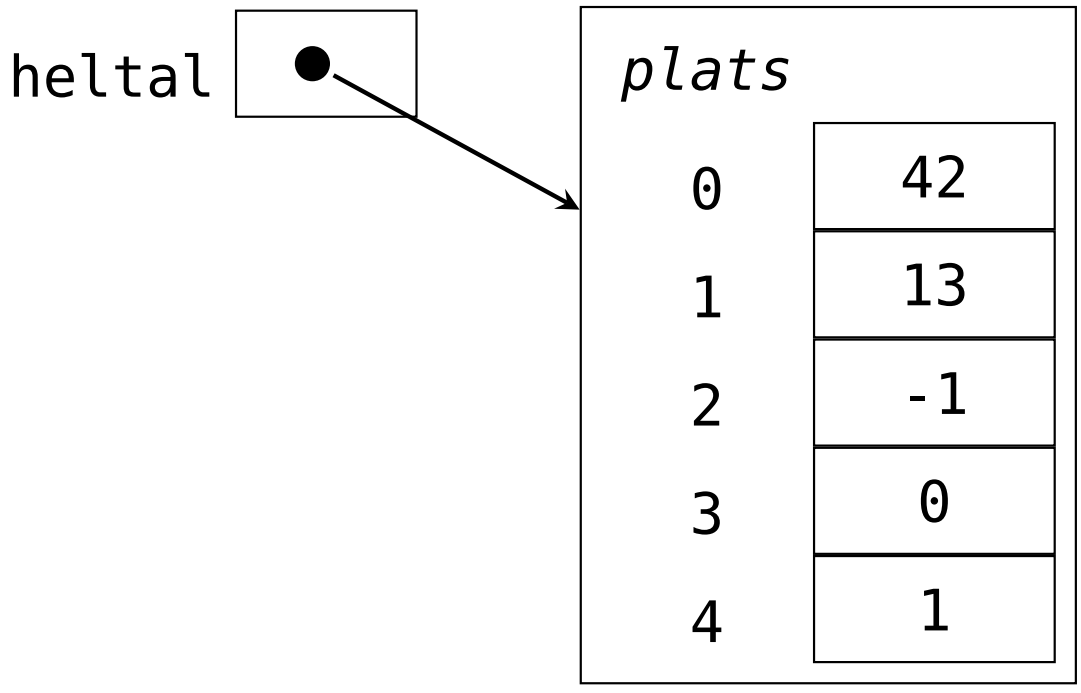
```
1  scala> val heltal = Vector(42, 13, -1, 0 , 1)
2  heltal: scala.collection.immutable.Vector[Int] = Vector(42, 13, -1, 0, 1)
3
4  scala> heltal.apply(0)
5  res0: Int = 42
6
7  scala> heltal(1)    // man kan skippa .apply
8  res1: Int = 13
9
10 scala> heltal(5)
11 java.lang.IndexOutOfBoundsException: 5
12   at scala.collection.immutable.Vector.checkRangeConvert(Vector.scala:132)
```

Utelämnar du `.apply` så gör kompilatorn anrop av `apply` ändå om det går.

[1] Vektor kallas ibland på svenska även fält, men det skapar stor förvirring eftersom det engelska ordet *field* ofta används för *attribut* (förklaras senare).

En konceptuell bild av en vektor

```
scala> val heltal = Vector(42, 13, -1, 0 , 1)
```



Elementen ligger på rad någonstans i minnet.

En samling strängar

- En vektor kan lagra många värden av samma typ.
- Elementen kan vara till exempel heltal eller strängar.
- Eller faktiskt vad som helst.

```
1  scala> val grönsaker = Vector("gurka","tomat","paprika","selleri")
2  grönsaker: scala.collection.immutable.Vector[String] = Vector(gurka, tomat, pa
3
4  scala> val g = grönsaker(1)
5  g: String = tomat
6
7  scala> val xs = Vector(42, "gurka", true, 42.0)
8  xs: scala.collection.immutable.Vector[Any] = Vector(42, gurka, true, 42.0)
```

Loopa genom elementen i en vektor

En **for-sats** som skriver ut alla element i en vektor:

```
1  scala> val grönsaker = Vector("gurka","tomat","paprika","selleri")
2
3  scala> for (g <- grönsaker) println(g)
4  gurka
5  tomat
6  paprika
7  selleri
```

Bygga en ny samling från en befintlig med for-uttryck

Ett **for-yield-uttryck** som **skapar en ny samling**.

```
for (g <- grönsaker) yield "god " + g
```

```
1  scala> val grönsaker = Vector("gurka","tomat","paprika","selleri")
2
3  scala> for (g <- grönsaker) yield "god " + g
4  res0: scala.collection.immutable.Vector[String] =
5    Vector(god gurka, god tomat, god paprika, god selleri)
6
7  scala> val åsikter = for (g <- grönsaker) yield s"god $g"
8  åsikter: scala.collection.immutable.Vector[String] =
9    Vector(god gurka, god tomat, god paprika, god selleri)
```

Samlingen Range håller reda på intervall

- Med en `Range(start, slut)` kan du skapa ett intervall: från och med `start` till (men inte med) `slut`

```
scala> Range(0, 42)
res0: scala.collection.immutable.Range =
  Range(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
    15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28,
    29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41)
```

- Men alla värden däremellan skapas inte förrän de behövs:

```
1 scala> val jättestortIntervall = Range(0, Int.MaxValue)
2 jättestortIntervall: scala.collection.immutable.Range = Range(0, 1, 2, 3, 4, 5
3
4 scala> jättestortIntervall.end
5 res1: Int = 2147483647
6
7 scala> jättestortIntervall.toVector
8 java.lang.OutOfMemoryError: GC overhead limit exceeded
```

Loopa med Range

Range används i for-lopar för att hålla reda på antalet rundor.

```
scala> for (i <- Range(0, 6)) print(" gurka " + i)
 gurka 0 gurka 1 gurka 2 gurka 3 gurka 4 gurka 5
```

Du kan skapa en Range med `until` efter ett heltal:

```
scala> 1 until 7
res1: scala.collection.immutable.Range =
  Range(1, 2, 3, 4, 5, 6)

scala> for (i <- 1 until 7) print(" tomat " + i)
 tomat 1 tomat 2 tomat 3 tomat 4 tomat 5 tomat 6
```

Loopa med Range skapad med `to`

Med `to` efter ett heltal får du en Range till och **med** sista:

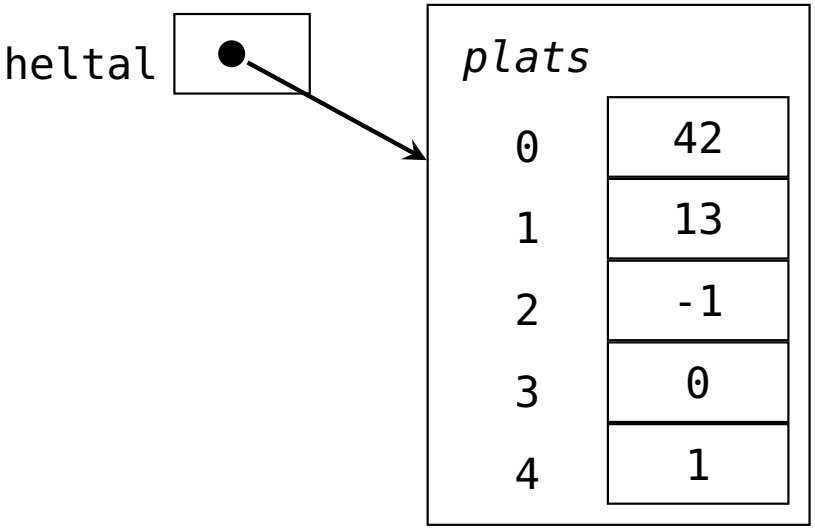
```
scala> 1 to 6
res2: scala.collection.immutable.Range.Inclusive =
  Range(1, 2, 3, 4, 5, 6)

scala> for (i <- 1 to 6) print(" gurka " + i)
 gurka 1 gurka 2 gurka 3 gurka 4 gurka 5 gurka 6
```

Vad är en `Array` i JVM?

- En `Array` liknar en `Vector` men har en särställning i JVM:
  - Lagras som en sekvens i minnet på efterföljande adresser.
  - **Fördel**: snabbaste samlingen för element-access i JVM.
  - Men det finns en hel del **nackdelar** som vi ska se senare.

```
scala> val heltal = Array(42, 13, -1, 0 , 1)
```



Huvudprogram med `main` i Scala och Java

Ett minimalt program i Scala och Java

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

```scala
// this is Scala

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

```java
// this is Java

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

Loopa genom en samling med en `while`-sats

Loopa genom argumenten i ett Scala-huvudprogram

Loopa genom argumenten i ett Java-huvudprogram

Algoritmer

Vad är en algoritm?

En algoritm är en sekvens av instruktioner
som beskriver hur man löser ett problem

Exempel:
matrecept

Vad är en algoritm?

En algoritm är en sekvens av instruktioner
som beskriver hur man löser ett problem

Exempel:
- matrecept
- uppdatera highscore i ett spel
- ...

XX

Algoritmexempel: PRODUKTEN AV n TAL

Funktioner och procedurer skapar struktur

Block

Kapslar in många satser.
Vad är ett block? I Scala (till skillnad från många andra språk) har ett block har ett värde

Block kan ha lokala variabler

Synlighetsregler etc

Funktioner

Parameter och argument

Procedurer

- En **procedur** är en funktion som **gör** något intressant, men som **inte** lämnar något intressant returvärde.
- Exempel på befintlig procedur: `println("hej")`
- Du **deklarerar egna procedurer** genom att ange **`Unit`** som returvärdestyp. Då ges värdet **`()`** som betyder ”inget”.

```
1  scala> def hej(x: String): Unit = println(s"Hej på dej $x!")
2  hej: (x: String)Unit
3
4  scala> hej("Herr Gurka")
5  Hej på dej Herr Gurka!
6
7  scala> val x = hej("Fru Tomat")
8  Hej på dej Fru Tomat!
9  x: Unit = ()
```

- Det som **görs** kallas (sido)**effekt**. Ovan är utskriften själva effekten.
- Funktioner kan också ha sidoeffekter. De kallas då **oäkta** funktioner.

Ingenting är faktiskt någonting i Scala

I många språk (Java, C, C) är funktioner som saknar värden speciella. I Java finns nyckelordet `void` men inte i Scala. I Scala är procedurer inte speciell efter som de faktisk returnerar ett värde som representerar ingenting, nämligen () som är av typen Unit. På så sätt blir procedurer inget undantag utan följer vanliga regler för syntax och semantik som för alla andra funktioner.

`https://en.wikipedia.org/wiki/Void_type`
`https://en.wikipedia.org/wiki/Unit_type`

Abstraktion: Problemlösning genom nedbrytning i enkla funktioner och procedurer som kombineras

- En av de allra viktigaste principerna inom programmering är **funktionell nedbrytning** där **underprogram** i form av funktioner och procedurer skapas för att bli byggstenar som kombineras till mer avancerade funktioner och procedurer.
- Genom de namn som definieras skapas **återanvändbara abstraktioner** som kapslar in det funktionen gör.
- Problemet blir med bra byggblock lättare att lösa.
- Abstraktioner som beräknar eller gör **en enda, väldefinierad sak** är enklare att använda, jämfört med de som gör många, helt olika saker.
- Abstraktioner med **välgenomtänkta namn** är enklare att använda, jämfört med kryptiska eller missvisande namn.

Exempel på funktionell nedbrytning

<<kvadrat, stapel, rutnät i kojo>>

Paket

<<paket skapar katalogstruktur för kodfiler>>

Jar-filer

<<liknar zip-filer; packar ihop bytekod i en enda fil för enkel distribution>>

Att göra i Vecka 2: Förstå grundläggande kodstrukturer

1. Laborationer är **obligatoriska**.
   Ev. sjukdom måste anmälas **före** till kursansvarig!
2. Gör övning `programs`
3. OBS! Ingen lab denna vecka w02. Använd tiden att komma ikapp om du ligger efter!
4. Träffas i samarbetsgrupper och hjälp varandra att förstå.
5. Om ni inte redan gjort det: Gör klart samarbetskontrakt och visa för handledare på resurstid.
6. **Koda på resurstiderna** och få hjälp och tips!