Datorlaboration 3

Måns Magnusson

January 17, 2014

Instruktioner

- Denna laboration ska göras **en och en**.
- Det är tillåtet att diskutera med andra, men att plagiera eller skriva kod åt varandra är **inte tillåtet**.
- Utgå från laborationsfilen som går att ladda ned **här**
- Laborationen består av två delar:
  - Datorlaborationen
  - Inlämningsuppgifter
- Innan du lämnar in laborationen:
  1. Starta om R-Studio eller rensa den globala miljön (Global environment) med `rm(list = ls())`.
  2. Ladda in funktionerna i R med `source`.
  3. Kontrollera att inget annat än funktionerna laddas in.
  4. Testa att funktionerna fungerar en sista gång.
- Deadline för labben framgår på **kurshemsidan**

Contents

I Datorlaboration **3**

1 Villkorssatser **3**

2 Loopar **4**
2.1 `for` - loopar 4
2.2 Nästlade `for`-loopar 4
2.3 `while` loopar 5
2.4 Kontrollstrukturer för loopar 6

II Inlämningsuppgifter **7**

Part I
Datorlaboration

1 Villkorssatser

1. Villkorssatser används för att kontrollera flödet i programmeringen på ett smidigt sätt. Skapa `if`-satsen nedan. Pröva att ändra värdet på `x` på lämpligt sätt och se hur resultatet av `if`-satsen ändras.

```
# if-sats:
x <- -1000
if (x < 0) print("Hej!")

[1] "Hej!"
```

2. För att kunna göra fler beräkningar i en `if`-sats måste { } användas. Kör koden nedan. Ändra värdet av `x`.

```
x <- -20
if (x < 0) {
    print("Positivt x")
    a <- pi + 23
    print(a)
}

[1] "Positivt x"
[1] 26.14
```

3. Testa nu att köra följande `if else`-sats (testa med olika värden för `x`)

```
if (x < 0) {
    print("Negativt x")
    a <- pi - 23
    print(a)
} else {
    print("Positivt eller noll")
    a <- pi + 23
    print(a)
}
```

4. Testa nu att köra en `if - else if - else` - sats med flera nivåer:

```
if (x == 0) {
    print("x <U+00E4>r noll")
} else if (x < 0) {
    print("x <U+00E4>r negativ")
} else {
    print("x <U+00E4>r positiv")
}
```

5. Skapa variabeln `cool_kvinna`. Skapa en `if - else if - else` sats som skriver ut födelseåret för följande kvinnor (och ta er lite tid att läsa om dem på Wikipedia):
   (a) Amelia Earhart
   (b) Ada Lovelace
   (c) Vigdis Finnbogadottir
   (d) Alice Walker
6. Gör om uppgift 5 men använd nu `switch()` istället för `if - else if - else`.

2 Loopar

2.1 `for` - loopar

1. En `for`-loop har ett loop-element och en loop-vektor. I koden nedan är `i` loop-elementet och `1:10` är vektorn som loopas över. Testa att köra koden. Testa att ändra `1:10` till `1:5` och `5:1`. Vad händer nu? Testa att använda loop-vektorn `seq(1, 6, by=2)`

```
for (i in 1:10) {
    x <- i + 3
    print(x)
}
```

2. Skriv en `for`-loop som skriver ut texten `Övning ger färdighet` 20 gånger.
3. Testa nu att köra koden nedan. Vad händer? Testa att ändra på vektorn `minVektor` till lämplia värden. Vilka värden ska `minVektor` ha om du vill bara skriva ut de tre sista orden?

```
minaOrd <- c("campus", "sal", "kravall", "tenta", "senare", "konjunktur")
minVektor <- 1:5
for (i in minVektor) {
    print(minaOrd[i])
}
```

4. Skriv en `for`-loop som skriver ut alla heltal som är jämt delbara med 13 som finns mellan 1 och 200. [**Tips!** `?%%`]
5. Skriv en `for`-loop som loopar över varje rad `i` matrisen `A` (se nedan). För varje rad ska medelvärdet beräknas. Skapa en text-sträng på formen `På rad i är medelvärdet x`, där `i` är radnummret och `x` är medelvärdet för just den raden. För att se om du gjort rätt: Rad 8 ska ha medelvärdet 23. [**Tips!** använd `paste()` för att skapa text-strängarna.]

```
A <- matrix(1:40, nrow = 10)
```

6. Kombinera en `for`-loop och villkorssats:
   Skriv en `for`-loop som skriver ut alla heltal som är jämt delbara med 3 som finns mellan 1 och 200. Förutom att skriva ut dessa tal ska de även sparas i en vektorn `delatMedTre`. Men bara de tal som är **udda** ska vara med. Använd en villkorssats för att göra den förändingen. Om ett av talen är jämt, så skriv ut texten `‘‘Intresserar mig inte’’` till skärmen. (**Tips!** `?%%`)

2.2 Nästlade `for`-loopar

1. Följande kod är ett exempel på en nästlad loop. Vad gör koden? Implementera denna kod.

```
A <- matrix(1:4, ncol = 2)
B <- matrix(5:8, ncol = 2)
C <- matrix(rep(0, 4), ncol = 2)
for (i in 1:2) {
    for (j in 1:2) {
        C[i, j] <- A[i, j] + B[i, j]
    }
}
```

2. Ändra koden ovan för matriser som är av storlek 3×3. Testa med följande två matriser:

```
A <- matrix(1:9, ncol = 3)
B <- matrix(10:8, ncol = 3)
```

3. Ändra koden igen koden för två godtyckligt stora matriser. [**Tips!** `dims()`]

2.3 `while` loopar

1. En `while`-loop loopar så länge villkoret är sant och inte ett bestämt antal gånger som `for`-loopar. Testa koden nedan med några olika värden på `x`.

```
x <- 1
while (x < 10) {
    print("x is less than 10")
    x <- x + 1
}
```

2. Om inte `while`-loopar skrivs på rätt sätt kan de loopa i "oändlighet". Vad är viktigt att tänka på i syntaxen när while-loop används för att undvika detta?
Obs! Om du testar koden nedan vill du nog avbryta.
I R-studio: trycka på stop-knappen i kanten på console - fönstret eller med menyn "Tools"-¿"Interrupt R".
Om du kör vanliga R: tryck "ctrl+C" .

```
x <- 1
while (x < 10) {
    print("x is less than 10")
    x <- x - 1
    print(x)
}
```

3. Skriv en `while` - loop som skriver ut alla jämna tal mellan 1 och 20.

4. I datorlaboration 1 skapades en funktion för att approximera talet e genom olika stora värden på N:

$$e = \sum_{n=0}^{N} \frac{1}{n!}$$

Skapa en variabel du kallar `tol` som du sätter till 0.001. Skapa en `while` loop som börjar vid $N = 1$ och ökar N med 1 till dess att skillnaden mellan approximationen av e och `exp(1)` är mindre än variabeln `tol`. Hur stort måste N vara för att felet på approximationen är mindre än 0.001, mindre än 0.00001?

2.4 Kontrollstrukturer för loopar

1. Nedan är ett exempel på kod som använder kontrollstrukturen `next`. Denna kontrollstruktur för loopar kan vara mycket bra för att hoppa över beräkningar senare i loopen. Det är ett sätt att innan loopen kör igång, pröva om denna iteration behöver eller ska beräknas.
Vad gör denna kod? Varför används `next()` här? Pröva att ta bort `next` och se vad som händer.

```
myList <- list("Hej", 3:8, "Lite mer text", "och lite nuffror", 4:12)
for (element in myList) {
    if (typeof(element) != "integer") {
        (next)()
    }
    print(mean(element))
}
```

2. I uppgift 4 on page 4 användes en for loop för att skriva ut alla tal som är delbara med 13 mellan 1 och 200. Använd nu `next` för att uppnå samma resultat.

3. På samma sätt som next kan användas för att begränsa vissa beräkningar kan break avsluta en for-loop när exempelvis en beräkning är tillräckligt bra. Det blir då en form av while loop fast med ett begränsat antal iterationer. `while` loopen i uppgift 1 på sida 5 kan på detta skrivas om med `break` på följande sätt. Pröva denna kod och experimentera lite med `x`. När vill du använda en `while`-loop och när en `for`-loop med `break`?

```
x <- 1
for (i in 1:10) {
    if (x < 10) {
        (break)()
    }
    print("x is less than 10")
    x <- x + 1
}
```

4. Pröva nu att på samma sätt konvertera din `while` loop i uppgift 4 på sida 5 till en `for`-loop med `break`.

Part II
Inlämningsuppgifter

Tips!

Inlämningsuppgifterna innebär att konstruera funktioner. Ofta är det bra att bryta ned programmeringsuppgifter i färre små steg och testa att det fungerar i varje steg.

Ett förslag på hur du kan angripa problemet är att:

1. Lös uppgiften med vanlig kod direkt i R-Studio (precis som i datorlaborationen ovan) utan att skapa en funktion.
2. Testa att du får samma resultat som testexemplen.
3. Implementera koden du skrivit i 1. ovan som en funktion.
4. Testa att du får samma resultat som i testexemplen, nu med funktionen.

Varje uppgift kan ge flera poäng så även om du inte lyckas med alla delar i en funktion kan du få poäng.

Uppgift 1: BMI

Skriv en funktion som du kallar `bmi()` med argumenten `bodyHeight` och `bodyWeight`. Funktionen ska beräkna BMI på följande sätt:

$$\texttt{bmi(bodyWeight,bodyHeight)} = \frac{\texttt{bodyWeight}}{\texttt{bodyHeight}^2}$$

och returnera värdet. Om `bodyLength` och/eller `bodyWeight` är mindre eller lika med 0 ska funktionen varna att den aktuella variabeln är mindre eller lika med 0 och att resultatet inte är meningsfullt:

`bodyWeight is not positive, calculation is not meaningful` eller
`bodyHeight is not positive, calculation is not meaningful`
Testa med olika värden för `bodyLength` och `bodyWeight`.

Här är ett textexempel på hur funktionen ska fungera:

```
bmi(bodyWeight = 95, bodyHeight = 1.98)

[1] 24.23

bmi(bodyWeight = 74, bodyHeight = -1.83)

Warning:  bodyHeight is not positive, calculation is not meaningful

[1] 22.1

bmi(bodyWeight = 0, bodyHeight = 1.63)

Warning:  bodyWeight is not positive, calculation is not meaningful

[1] 0

bmi(bodyWeight = -73, bodyHeight = 0)

Warning:  bodyWeight is not positive, calculation is not meaningful
Warning:  bodyHeight is not positive, calculation is not meaningful

[1] -Inf
```

Uppgift 2: Matrismultiplikation

En central del inom den linjära algebran är matrismultiplikation, d.v.s. att multiplicera två matriser med varandra. Du ska nu skapa en funktion kallad myMatrixProd() med argumenten A och B som multiplicerar två matriser med varandra på följande sätt:

$$\texttt{myMatrixProd(A, B)} = \mathbf{A} \cdot \mathbf{B}$$

Om dimensionerna inte gör det möjligt att multiplicera matriserna ska funktionen stoppas och returnera felmeddelandet `Matrix dimensions mismatch`. Observera att det inte är tillåtet att använda R:s funktion för matrismultiplikation `%*%`. Du får dock använda den för att generera fler testfall för att testa att din funktion räknar rätt.

De steg funktionen kan gå igenom är följande:

1. Pröva om dimensionerna av matris **A** och **B** innebär att de kan multipliceras med varandra, stoppa annars funktionen och returnera felmeddelandet.
2. Skapa en ny matris (ex. kallad **C**) med de dimensioner som produkten av A och B har.
3. Loopa över elementen i **C** och räkna ut varje element för sig. [**Tips!** Här kan du använda din kod från uppgift 3 i datorlaboration 1]

Här är textexempel på hur funktionen ska fungera:

```
matX <- matrix(1:6, nrow = 2, ncol = 3)
matY <- matrix(6:1, nrow = 3, ncol = 2)
myMatrixProd(A = matX, B = matY)

     [,1] [,2]
[1,]   41   14
[2,]   56   20

myMatrixProd(A = matY, B = matX)

     [,1] [,2] [,3]
[1,]   12   30   48
[2,]    9   23   37
[3,]    6   16   26

myMatrixProd(A = matX, B = matX)

Error :  Matrix dimensions mismatch
```

Uppgift 3: Den babylonska metoden för att approximera kvadratrötter

En algoritm för att approximera kvadratroten ur ett tal är den så kallade babylonska metoden, en metod som flera säkert känner igen från gymnasiet. Metoden, som är ett specialfall av Newton-Raphsons metod, kan beskrivas på följande sätt:

1. Starta med ett godtyckligt förslag på kvadratroten, kallat r_0
2. Beräkna ett nytt förslag på roten på följande sätt:

$$r_{n+1} = \frac{r_n + \frac{x}{r_n}}{2}$$

3. Om $|r_{n+1} - r_n|$ inte har uppnått godtycklig nogrannhet: gå till steg 2 igen.

Implementera denna algoritm som en funktion i R. Funktionen ska heta `babylon()` och argumenten `x`, `init` och `tol`. `x` är talet för vilket kvadratroten ska approximeras, `init` är det första förslaget på kvadratroten och `tol` är hur stor noggrannhet som ska krävas.

Funktionen kan implementeras antingen som en `for` - loop med `break` eller en `while` loop. Funktionen ska returnera en lista med två element, `rot` och `iter` (båda numeriska värden). I elementet `rot` ska

approximationen av kvadratroten returneras och i elementet `iter` ska antalet iterationer returneras.

Här är textexempel på hur funktionen ska fungera:

```
babylon(x = 2, init = 1.5, tol = 0.01)

$rot
[1] 1.414

$iter
[1] 2

sqrt(2)

[1] 1.414

babylon(x = 3, init = 2, tol = 1e-06)

$rot
[1] 1.732

$iter
[1] 4

sqrt(3)

[1] 1.732
```

Nu är du klar!