

Datorlaboration 1

Måns Magnusson

January 8, 2014

Instruktioner

- Denna laboration ska göras **en och en**.
 - Det är tillåtet att diskutera med andra, men att plagiera 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 "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	R som miniräknare, kod-filer och variabler	3
2	Global enviroment och hjälpen	4
3	Variabler, <code>print()</code> , <code>cat()</code> och vektorer (forts.)	4
4	Introduktion till funktioner i R och <code>source()</code>	7
II	Inlämningsuppgifter	9

Part I

Datorlaboration

1 R som miniräknare, kod-filer och variabler

1. Öppna R-Studio.
2. Gör följande beräkningar:

```
> 3 + 4
> (5 * 6)/2
> 45 - 2 * 3
> (45 - 2) * 3
> 3^3
> sqrt(4)
> exp(1)
> abs(-3)
> 7%%3
> pi
> sin(pi)
> cos(pi)
> tan(pi)
```

3. Gör följande beräkning:

```
> sqrt(abs(-3)^2 - 3)
```

4. Prova att gör beräkningarna med variabler istället. [**Tips:** Variablerna blir synliga under “Enviroment”-fliken i R-Studio.]:

```
> a <- -3
> b <- 2
> c <- sqrt(abs(a)^b + a)
> c

[1] 2.449
```

5. Får du samma resultat som under ovan?
6. Starta om R-Studio . Finns variablerna kvar i workspace? [**Obs:** R-Studio kommer fråga om du vill spara variablerna i ditt workspace. Svara “Don’t save” på denna fråga.]
7. Skapa en ny R - fil. [**Tips:** File → New file... → R Script].
8. Gör följande beräkning i denna fil och spara x som variabel i R-filen.

$$x = \sqrt{z^2 + |y|}$$

$$\text{där } z = e^{1+\frac{3}{13}} - 1 \text{ och } y = \ln\left(\frac{\pi}{17}\right)$$

Exempel på hur z kan beräknas finns nedan. Beräkna y och x :

```
> z <- exp(1 + 3/13) - 1
```

9. För att kommentera sin kod används # som kan användas för att kommentera en hel rad (eller resten av en rad). Allt efter symbolen (till nästa rad) körs inte av R. Prova att skriva kommentaren:

```
> # My first comment
```

10. Prova att spara ned din fil som myFirstRScript.R [Tips: File → Save as...].

2 Global enviroment och hjälpen

1. I R sparas alla variabler i datorns interna minne (RAM). Detta kallas för R:s “Global enviroment”. För att undersöka vilka variabler du har i Global enviroment används funktionen ls() [Tips: I R-Studio är det möjligt att se Global enviroment direkt i fliken “Enviroment”].
2. Genomför följande kommandon för att se vilka objekt som finns i Gobal enviroment och ta bort objektet a:

```
> a <- c(1, 5, 2)
> ls()

[1] "a" "b" "c"

> rm(a)
> ls()

[1] "b" "c"
```

3. Ta bort (radera) alla variabler i Global enviroment med funktionen rm().
4. Hur ser workspacet ut? Använd funktionen ls().
5. Vad innebär exakt funktionerna ls() och rm()? Använd hjälpen för att läsa om funktionerna. [Tips: pröva help(ls) och help(rm)]

3 Variabler, print(), cat() och vektorer (forts.)

1. Skapa numeriska och textvariabler och undersök vad variabelernas typ heter i R. [Tips: ?mode]

```
> minNum <- 2013
> minText <- "Mer R till studenterna"
>
> print(minNum)

[1] 2013

> print(minText)

[1] "Mer R till studenterna"

>
> # Exempel
> mode(minNum)

[1] "numeric"
```

2. Vilken typ (mode) av variabel är `minText`? [**Tips:** `?mode`]
3. Prova följande kod och diskutera resultatet. Vad är skillnaden mellan den första och tredje raden?

```
> a <- 5
> a

[1] 5

> a < -7

[1] FALSE

> a

[1] 5
```

4. Läs hjälpen till funktionen `print()` och prova följande kod:

```
> x <- "The value of pi is"
> print(x)

[1] "The value of pi is"

> print(pi)

[1] 3.142
```

5. Upprepa koden ovan, men byt ut `print` mot `cat`. Med hjälp av `cat()`, Skriv ut följande text på skärmen:

```
The value of pi is: 3.142
```

6. Skapa följande vektorer:

```
> myVar1 <- c(1, 2, 3, 4, 5)
> myVar2 <- c("Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun")
> myVar3 <- c(0.2, 0.4, 0.6, 0.8, 1)
> myVar4 <- c("Jim", "Apple", "Linda", "School", "Math")
> myVar5 <- c(2, 3, 6, 7, 8, 9)
```

7. Hur många element finns i varje vektor [**Tips:** `?length`]
8. Vad är det för typ av vektor? [**Tips:** `?mode`]
9. Gör följande beräkningar:

```
> myVar1 + myVar3

[1] 1.2 2.4 3.6 4.8 6.0
```

```
> myVar1 - myVar3

[1] 0.8 1.6 2.4 3.2 4.0

> myVar1 * myVar3

[1] 0.2 0.8 1.8 3.2 5.0

> myVar1/myVar3

[1] 5 5 5 5 5

> (myVar1 - myVar3)

[1] 0.8 1.6 2.4 3.2 4.0

> myVar1 - exp(myVar3)

[1] -0.2214 0.5082 1.1779 1.7745 2.2817

> log(myVar1)

[1] 0.0000 0.6931 1.0986 1.3863 1.6094

> log(myVar1) + 1

[1] 1.000 1.693 2.099 2.386 2.609
```

10. Beräkningarna som gjordes ovan sker element för element. Men det finns också funktioner för att beräkna summan (`sum()`), medelvärdet (`mean()`), standardavvikelsen (`sd()`), kvantiler (`quantile()`) m.m. för elementen i en vektor. Gör följande beräkningar:

```
> mean(myVar1)

[1] 3

> sum(myVar3)

[1] 3

> sd(myVar3)

[1] 0.3162

> quantile(myVar3)

 0%  25%  50%  75% 100%
0.2  0.4  0.6  0.8  1.0
```

11. Skapa vektorn `x` på följande sätt..

```
> # Skapa vektorerna myVar1 och myVar4 (om inte redan gjort)
> myVar1 <- c(1, 2, 3, 4, 5)
> myVar4 <- c(0.2, 0.4, 0.6, 0.8, 1)
> # Skapa vektorn x
> x <- c(myVar1 + myVar4, myVar4)
```

- Hur många element har vektorn `x`. [Tips: `?length`]
- Vad är medelvärde för vektorn `x`.

12. Skapa följande vektorer i R.

$$k = (12, \pi, 1, 7), l = (2 \cdot \sqrt{1}, 2 \cdot \sqrt{2}, 2 \cdot \sqrt{3}) \text{ och } m = (e, \ln(2 + e))$$

4 Introduktion till funktioner i R och `source()`

- Skapa en ny R-fil med namnet `minaFunktioner.R`. Vi ska nu göra en fil som med funktioner.
- Som nämnts tidigare består en funktion av
 - Ett funktionsnamn (ex. `minFunktion`) som "tillskrivs" en funktion
 - En funktionsdefinition - `function()`
 - Noll eller flera argument (ex. `x, y`)
 - "Curly Bracers" som "innehåller" funktionen `{}`
 - Beräkningar / programkod (ex. `x+y`)
 - Returnera argument med `return()`
 - Exempel på funktion:

```
> minFunktion <- function(x, y) {
+   z <- x + y
+   return(z)
+ }
```

- Det kan vara svårt att få till en hel funktion direkt. Det bästa är att skapa funktionen i flera steg:

```
> # Steg 1: Skriv koden och testa att den fungerar
> x <- 3
> y <- 5
> z <- x + y
> z

[1] 8

> # Steg 2: Lyft in koden (som du vet fungerar) i funktionen:
> minFunktion <- function(x, y) {
+   z <- x + y
+   return(z)
+ }
> # Steg 3: Ta bort x, y, z i Global environment (mer om anledningen kommer
> # senare)
> rm(x, y, z)
>
> # Steg 4: Testa att funktionen fungerar
> minFunktion(x = 3, y = 5)
```

```
[1] 8

> # Yay! Det funkar!
```

4. Skriv in funktionen ovan i R. Denna kan beskrivas matematiskt som:

$$\text{minFunktion}(x, y) = x + y$$

Pröva funktionen med olika värden på argumenten x och y . När du kör funktionen, skapas variabeln z i “Global enviroment”? Varför inte?

```
> minFunktion(3, 5)

[1] 8
```

5. Skriv in följande funktion i R. Vad gör den?

```
> nyFun <- function() {
+   vec <- c(1, pi, pi^2)
+   return(vec)
+ }
```

6. Skapa följande funktion i R

$$f(x) = x^2 + \sin(x \cdot \pi)$$

7. Skapa följande funktion för skalär multiplikation i R, där \mathbf{a} är en vektor av godtycklig längd. (Om du inte vet hur skalär multiplikation görs med en vektor finns information [\[här\]](#))
Såhär kan funktionen beskrivas matematiskt:

$$g(\mathbf{a}, b) = b \cdot \mathbf{a}$$

där \mathbf{a} är en vektor.

8. Skapa en funktion **utan argument** som skriver ut ‘‘Hello World!’’ till skärmen. [Tips! pröva både `cat()` och `print()`]
9. Spara ned din R-fil med bara funktionerna. Ta bort eventuell kod som inte är en del av funktionerna. Starta om R-Studio eller rensa “Global enviroment” genom att klicka på “Clear” under fliken “Enviroment”.
10. Ofta vill man köra en hel fil med kod på en gång, exempelvis om man har skapat flera funktioner som du vill läsa in i en R session. För detta används funktionen `source()`. Uppe till höger i source-fönstret i R-Studio finns en knapp där det står “source”.

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 (interaktivt) med vanlig kod direkt i R-Studio (precis som i datorlaborationerna ovan).
2. Testa att du får samma resultat som testexemplen.
3. Implementera koden du skrivit i inlämningsfilen 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. Det viktigaste är att funktionen fungerar och returnerar ett värde.

Uppgifter

1. Skapa en funktion som heter **uppgift1** utan argument. Funktionen ska beräkna och returnera följande vektor med tre element i R. **3p**

$$(\ln 3, e^{\pi+1}, \sin(\frac{\pi}{3}))$$

I exemplet nedan har värdena avrundats till två decimaler. Observera att för att funktionen ge full poäng ska resultatet stämma med fler decimaler.

```
> uppgift1()
[1] 1.0986 170.9879 -0.3272
```

2. Skapa en funktion som heter **uppgift2** med argumentet **vektor**. Funktionen ska returnera produkten av det första och sista elementet i **vektor**.

```
> uppgift2(vektor = c(3, 1, 12, 2, 4))
[1] 12

> uppgift2(vektor = c(3, 1, 12))
[1] 36
```

3. Skapa en funktion som heter **uppgift3** som beräknar skalärprodukten mellan två vektorer, **a** och **b**. Skalärprodukten beräknas på följande sätt:

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^n a_i b_i = a_1 b_1 + a_2 b_2 + \dots + a_n b_n$$

Mer information om skalärprodukten finns här.

```
> uppgift3(a = c(3, 1, 12, 2, 4), b = c(1, 2, 3, 4, 5))

[1] 69

> uppgift3(a = c(-1, 3), b = c(-3, -1))

[1] 0
```

4. Skapa en funktion `uppgift4` som tar argumentet `namn`. Funktionen ska skriva ut `[namn]`, `I am your father.`, där `[namn]` ersätts med det värdet på argumentet `namn`. (OBS! använd `cat()`, inte `return()`) [Tips! `sep`]

```
> uppgift4(namn = "Luke")

Luke, I am your father.
```

5. Talet e kan beskrivas som följande oändliga serie:

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

Denna serie gör att talet e kan approximeras godtyckligt bra på följande sätt, genom att välja ett värde på N :

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

Skapa en funktion i R som du kallar `uppgift5` med argumentet `N` för att skapa en godtyckligt noggrann approximation av e . Prova hur stort `n` behöver vara för att funktionen ska approximera e till och med fjärde decimalen. [Tips! för att få e i R, använd `exp(1)`]

```
> uppgift5(N = 2)

[1] 2.5

> uppgift5(N = 4)

[1] 2.708
```

Nu är du klar!