Datorlaboration 6

Josef Wilzén

25 februari 2014

Instruktioner

- Denna laboration ska göras i **grupper om två personer**. Det är viktigt att att följa gruppindelningen och inte ändra grupper. Om det är problem med grupperna så ska ni prata med Josef eller Måns.
- Det är tillåtet att diskutera med andra grupper, 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**

Innehåll

Parprogrammering

Tanken är att ni ska öva på parprogrammering under labb4 till labb8.

- Detta innebär att två personer samarbetar och tillsammans löser programmeringsproblem vid en dator.
- Personerna turars om att ha rollerna:
  - **Föraren:** har kontroll över tangentbordet och skriver koden
  - **Navigatören:** Är delaktig i problemt genom att kommentera, diskutera och analysera koden som skrivs. Den här personen ska **inte** sitta och titta passivit.
- Det är viktigt att byta roller **ofta** och att båda personerna är involverade i lösningen av problemet. Vi rekomenderar att ni byter roller minst varje 30 min.
- Det är viktigt att kommentera sin kod så att ni båda kan förstå koden i efterhand. Tänk på skriva ner syftet med koden och inte exakt vad koden gör.
- Syftet med parprogrammering är:
  - Lära av varandra
  - Lära sig olika sätt att skriva kod (stilar) och olika sätt att lösa problem
  - Skriva mer effektiv kod med mindre buggar
  - Lära sig skriva kod som är lätt att förstå för andra

Del I
Datorlaboration

1 R-paket

1. Lista alla R-paket som finns installerade på din dator.
2. Lista samtliga R-paket som är inlästa i din R-session.
3. Ladda in paketet `stringr`.
4. Använd `help.search()` för att hitta generell information om paketet. Testa även `ls("package:stringr")` och `packageDescription()`. Leta reda att på den funktion som används för att räkna ut längden på en sträng (i antalet tecken).
5. Använd `example()` för att köra exemplen för denna funktion.
6. Ta bort paketet `stringr` från R:s minne.
7. Skapa funktionen `mad()` enligt koden nedan. Kör funktionen så den finns i R:s minne. Kontrollera att funktionen är inäst med funktionen `ls()`. Det finns en annan funktion med samma namn sedan tidigare i paketet `stats` som beräknar "median absolute deviation". Paketet i `stats` laddas alltid automatiskt in när R startas. Hur kan du anropa funktionen `mad()` i stats-paketet utan att döpa om eller ta bort din tidigare funktion?

```
mad <- function(x) {
    print("Detta är min mad-funtion")
    y <- sum(abs(x - median(x)))
    return(y)
}
```

8. Med funktionen `library()` är det bara möjligt att ladda in ett R-paket i R åt gången. Du vill skapa en funktion som kan läsa in flera R-paket direkt. Skapa en funktion som heter `CheckAndLoad()`. Denna funktion ska kunna ta en character-vector med paketnamn (ex. `c(''stringr'',''lubridate'')`) som argument. För varje textelement (eller paketnamn) ska funktionen undersöka om paketet finns installerat [Tips! Se uppgift 1]. Om paketet finns installerat ska funktionen ladda in paketet i R och skriva ut texten "R-Paketet [namnet på paketet] är nu inläst." till skärmen. Om paketet inte finns installerat i R ska funktionen skriva ut "R-paketet [namnet på paketet] är tyvärr inte installerat och kan inte läsas in." [Tips! `paste()` eller `str_c()` i stingr-paketet]
9. Om du har egen dator: Testa att installera paketen `stringr` och `lubridate` på din egna dator med `install.packages()`.

2 Slumptal och statistik

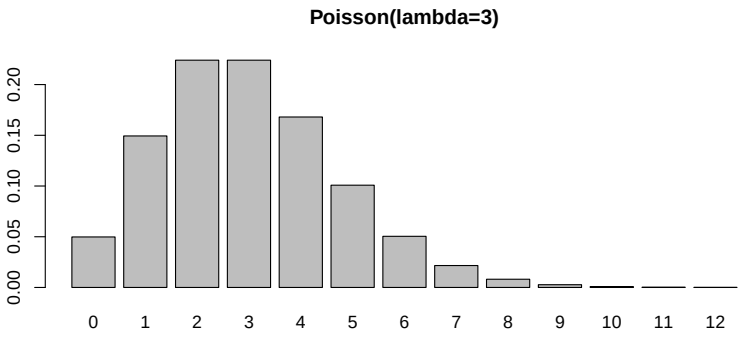
1. Skapa en vektor med slumptal av längd 30 från den uniforma fördelningen $U(min = 1, max = 10.2)$. Tips: `?distribution`

2. Skapa en vektor med slumptal av längd 200 från normalfördelningen, medelvärde=1 och varians=2, döp den till `minNorm`

3. Skapa ett histogram över `minNorm` och en boxplot

4. Skapa en vektor med slumptal av längd 50 från Poissonfördelningen med medelvärde 8, döp den till `minPoission`

5. X är poissonfördelad med medelvärde 3. Er uppgift är nu att skapa ett stapeldiagram(se figuren nedan) över sannolikheterna att X antar värdena $0, 1, ..., 11, 12$. Tips: `?dpois`



6. Låt X vara samma som i uppgift (5). Vad är sannolikheten att X antar följande värden:

   (a) Sitt medelvärde?

   (b) Mindre än eller lika med sitt medelvärde?

   (c) Större än sitt medelvärde?

   (d) Ett udda tal? Ni behöver inte ta hänsyn till tal som är större än 12.

   (e) Talen 4 till 6?

7. Skapa en vektor med slumptal av längd 50 från t-fördelningen, med medelvärde 1 och 5 frihetsgrader (degrees of freedom), döp den till `minT`

8. Gör ett histogram över `minPoission` och över `minT`. Testa att ändra "`breaks= `" i `hist()` till några olika värden.

9. Dra stickprov utan återläggning från sekvensen `1:10`

10. Upprepa (9). men med återläggning, vad blir skillnaden?

11. Tänk dig att du och några vänner ska organisera en fest. Skapa vektorn `namn`, som ska innehålla namnen för minst tre personer som strängar. Två personer behövs för att laga mat och två för att diska. Välj slumpmässigt vilka som ska göra de olika uppgifterna. Samma person ska kunna bli vald för båda uppgifterna.

12. Upprepa (11) två gånger, blir resultatet det samma? Upprepa två gånger till men kör först `set.seed(1234)` innan varje gång. Vad händer? (tips: `?set.seed()`)

13. Upprepa nu (11), men ändra argumentet "`prob=`" så att de olika personer har olika sannolikhet att bli vald.

14. Nu ska ni testa att simulera olika tärningskast. I uppgiften betyder D6 en vanlig 6-sidig tärning (med sidorna 1,2,3,4,5,6), och där alla utfall har samma sannolikhet (1/6).

    (a) Skriv en funktion `myDice(),` som generar n stycken kast från en D6 och returner en vektor med kasten. Tips: ?sample

    (b) Skriv en funktion `sumOfDice()` som generar följande slumptal: $Y_k = \sum_{n=1}^{N} X_n$, där X är ett tärningskast från funktionen `myDice()`, N är ett heltal, k går från $1, 2, 3, ..., K$. Funktionen ska ha N och K som argument, `sumOfDice(N=,K=)`. Funktionen ska alltså kasta N stycken D6, summera dessa, sen upprepa det K stycken gånger. Tips: skapa en vektor med K stycken nollor, fyll den sen med lämpliga värden i en for-loop.

    (c) Testa nu `sumOfDice(N=, K=)` med $N = 3, 5$ och $K = 100, 1000, 3000$. Plotta resultaten i histogram och beräkna också medelvärde, standardavvikelse, min och max. Nedan ser ni några exempel på tester:

```
set.seed(3827)
x <- sumOfDice(N = 3, K = 100)
x[1:10]

 [1]  8  7 11  8 16 14 11 12 14 10

set.seed(227)
x <- sumOfDice(N = 5, K = 1000)
x[1:15]

 [1] 14 20 12 14 17 19 17 13 15 15 10 19 16 10 16
```

15. Skapa funktionen `expectedValue(x=,px=)`, som givet vektorn x (innehåller numeriska värden) och px (innehåller sannolikheter för värdena i x) ska

beräkna det teoretiska väntevärdet: $E[X] = \sum_{i=1}^{K} x_i p_i(X = x)$, där index i går över alla möjliga värden på x. Ex: Om X följer fördelningen i tabellen nedan, då blir väntevärdet: $0 \cdot 0.3 + 1 \cdot 0.1 + 2 \cdot 0.6 = 1.3$

x	p(x)
0	0.3
1	0.1
2	0.6

16. Skapa funktionen `variance(x= px=)`, som givet vektorn x (innehåller numeriska värden) och px (innehåller sannolikheter för värdena i x) ska beräkna det teoretiska variansen. Titta här för att om du inte vet hur variansen beräknas.

17. Testa nu `expectedValue()` och `variance()` på fallen nedan.

```
x1 <- 1:50
px1 <- 1:50/sum(1:50)
x2 <- 1:30
px2 <- (sin(x2) + 1)/sum(sin(x2) + 1)
```

18. På hur många sätt kan du välja 3 element från en grupp av 6 element om ordningen inte spelar roll? Vilka är de olika kombinationerna? Tips: `choose(), combn()`

19. Skriv en egen funktion som givet n element beräknar alla permutationer som kan göras för k element.

3 HUS-data

1. Ni ska nu analysera datamaterialet HUS, som innehåller information om en mängd hus. Följande variabler finns i data:

   - Försäljningspris (dollar)
   - Bostadsyta (kvadratfot)
   - Antal sovrum
   - Antal badrum
   - Förekomst av luftkonditionering, 1 = luftkonditionering finns, 0 annars
   - Antal bilar som garaget är konstruerat för
   - Förekomst av pool, 1 = pool finns, 0 annars
   - Byggår

- Tomtstorlek (kvadratfot)

2. Läs in datamaterialet "HUS.csv" och spara det som `HUS`. Det finns även ett dataset "HUS_eng.csv", som har engelska namn på variablerna men innehåller samma data.

3. Gör följande med `HUS`

   (a) Plotta en boxplot över variabeln `Försäljningspris.` Ta fram besrkivande statistik med `summary()` för `Försäljningspris.` Det verkar finnas en del outliers (extremt stora värden), dessa vill vi ta bort från data, kör följande kod:

   ```
   # ta bort alla hus som har pris större än 3:e kvartilen:
   index <- HUS[, 1] < quantile(HUS[, 1])[4]
   HUS <- HUS[index, ]
   ```

   (b) Plotta ett histogram för `Försäljningspris` med `breaks=40`, (bonusfråga: ser fördelningen symmetrisk ut?)

   (c) Beräkna ett tvåsidigt konfidensintervall (KI) med $\alpha = 0.05$ för `Försäljningspris` (tips `?t.test()`) spara i `testPris.`

   (d) Välj ut KI från `testPris`. tips: `?t.test()` och läs under rubriken "Value". Kör sedan `str(testPris).` Testa `class(testPris)`

   (e) Beräkna ett tvåsidigt KI med $\alpha = 0.10$ för `Försäljningspris`

   (f) Beräkna ett enkelsidigt (undre) KI med $\alpha = 0.01$ för `Försäljningspris`

   (g) Kör koden nedan, vad blir resutlatet? Hur ska medelvärdet för variabeln `Luftkonditionering` tolkas?

   ```
   summary(HUS)
   ```

4. Skapa följande frekvenstabeller från HUS (tips: `?table()`)

   (a) För `Antal.sovrum`

   (b) För `Luftkonditionering`

   (c) Mellan variablerna `Antal.sovrum` och `Luftkonditionering`, spara som `sovLuftTab.` Testa `class(sovLuftTab)`

   (d) Mellan variablerna `Antal.sovrum` och `Antal.badrum`

5. Kör `prop.table(sovLuftTab)` och `prop.table(sovLuftTab,margin=1)`. Vad händer med tabellen?

6. Beräkna fishers exakta test för tabellen `sovLuftTab`, använd $\alpha = 0.05$, vad är noll-hypotesen? Kan vi förkasta den? (tips: `?fisher.test()`)

7. Antag att variablen `Försäljningspris` representerar en hel population med hus. Dra ett slumpmässigt stickprov med 20 hus utan återläggning. Beräkna ett tvåsidigt KI ($\alpha = 0.01$) utifrån stickprovet för populationsmedelvärdet.

8. Antag nu att hela datasetet HUS är alla hus i en population. Dra ett stickprov slumpmässigt (dvs välj rader) på 40 hus utan återläggning. Gör följande beräkningar utifrån stickprovet:

   (a) Numeriska variaber: Beräkna ett tvåsidigt KI med $\alpha = 0.01$för populationsmedelvärdet.

   (b) Binära(0/1) variaber: Beräkna ett tvåsidigt KI med $\alpha = 0.01$för populationsandelen. Tips: `?prop.test(), ?table()`

4 Datum och tid med lubridate

1. Ladda in paktet `lubridate` i den aktuella R-sessionen.

2. Spara ditt fördelsedatum som ett datum i R. Tips: `?ymd()`

3. Skapa ett intervall-objekt `myTime` som börjar vid din födelsedag och slutar idag.

4. Räkna ut följande:

   (a) Hur många dagar som finns i `myTime`

   (b) Hur många veckor som finns i `myTime`

   (c) Hur många år som finns i `myTime`

5. Skapa datumet "2010-04-23 12:33:45" med funktionen ymd_hms() och döp den till `testTime`. Gör följande beräknigar:

   (a) Välj ut året med year()

   (b) Välj ut dagen med day()

   (c) Välj ut timmen med hour()

   (d) Välj ut sekunden med second()

   (e) Kolla i artikeln om lubridate hur ni kan göra avrundningar under sektion 6. Avrunda till:

      i. Nedåt till år

      ii. Uppåt till dag

      iii. Närmste heltalsminuten

   (f) Ändra nu följande saker i testTime.

      i. Året till 1876

ii. Sekunden till 21
iii. Månaden till september.

6. Skapa nu fyra variabler: En som är en instant, en som är av typen interval, en av typen duration och en av typen period. Ni bestämmer själv vilka datum som variablerna ska innehålla. Kolla i föreläsningsanteckningarna hur du kan göra det. Testa sen att göra minst tre av de beräkningar som finns beskrivna i tabell 6 i artikeln om lubridate.

7. Skapa följande sekvenser: tips `?seq.Date()`

   (a) Alla dagar mellan 2014-01-20 till 2014-03-28
   (b) Varanan dag mellan 2014-01-20 till 2014-03-28, med start på den första dagen
   (c) Med datumet för alla fredagar under 2014
   (d) Med datumen för var fjärde vecka under hela 2014, med start i 2014-01-01.

8. Läs in datasetet "google.csv"från labb 2 i R och döp den till google.

9. Plotta variabeln `Close` mot dess index, dvs plot(google$Close)

10. Konvertera variabeln `Date` till en datumvariabel. Tips: `?as.character()`, `?ymd()`

11. Plotta variabeln `Close` mot variablen `Date`. Hur ser x-axeln ut?

12. Ett sätt att styra x-axelns utseende är med koden nedan, kör den och kolla vad som händer. Vad är skillnaden från plotten i (11)? Testa att ändra det andra argumentet i `pretty_dates()`, vad händer? Kolla i hjälpen för `pretty_dates()`.

```
plot(google$Date, google$Close, xaxt = "n")
temp <- pretty_dates(google$Date, 10)
axis.POSIXct(side = 1, at = temp)
```

13. Använd `table()` för att räkna ut hur många observationer som det finns...

   (a) För varje år i `Date`.
   (b) För varje månad i `Date`.
   (c) För varje vecka (nummer) i `Date`.
   (d) För varje veckodag i `Date`.
   (e) Gör en korstabell mellan år och månader

14. Räkna ur medelvärdet för `Close` i google för varje veckodag. Tips: `by()`

15. Gör ett histogram för varje månad för `Close` i `google`.

Del 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 uppgift kan du få poäng.

5 Gruppvisa konfindesintervall (5 p)

Nu ska ni skapa en funktion som ska beräkna gruppvisa konfindesintervall (KI) för en variabel. Innan ni börjar se till att HUS-data är inläst och kör koden nedan för att ta bort de extremt stora värdena:

```
# ta bort alla hus som har pris större än 3:e kvartilen:
index <- HUS[, 1] < quantile(HUS[, 1])[4]
HUS <- HUS[index, ]
```

Funktion ska heta `myGroupedTest(dataVector,myGroups,alpha)` och ha argumenten:

- `dataVector` - är en numerisk vektor
- `myGroups` - är en factor/character-vektor som grupperar dataVector
- `alpha` - är signifikansnivån för intervallet, alph=0.05 % ska ge ett 95 % KI.

Funktionen ska returnera en matris `result` där raderna motsvarar grupperna i `myGroups` och har fyra kolumner: Undre gräns för KI, medelvärdet, övre gräns för KI och antal observationer i varje grupp. Se testfallen för namen på kolumnerna. Raderna ska ha samma namn som grupperna `myGroups`.

Förslag till lösning:

1. Se till att myGroups är en faktor. Räkna ut hur många grupper som finns i myGroups. Tips: `?levels() ?table()`
2. Skapa en tom matris av rätt storlek, kalla den `result`. Ge den lämpliga namn. Tips: `?colnames(), ?rownames()`
3. Spara antalet observationer för varje grupp i den fjärde variabeln i `result`.
4. Använd `by()` kombinerat med `t.test()` för att beräkna gruppvisa KI, spara i `groupTest`. Testa `?by()`, ni ser att det finns ett argument som heter "`...`" för funktionen `by()`. Dessa tre punkter kan ersättas med argument som behövs till funtionen "`FUN`". Mer tips: `str("objekt från t.test")` och `?by` läs under rubriken "value" för att kolla vad `by()` returnerar.
5. Loopa över antalet grupper och välj ut KI och medelvärde för varje grupp från `groupTest`. Spara på rätt ställen i `result`.
6. Returnera `result`.

Testa om testfallen nedan fungerar:

```
# testar på HUS-data::
myGroupedTest(HUS[, 1], HUS$Luftkonditionering, 0.01)

  Lower CI-limit   Mean Upper CI-limit No of obs.
0         161468 173473         185479         82
1         213182 220556         227930        308

myGroupedTest(HUS[, 2], HUS$Pool, 0.1)

  Lower CI-limit Mean Upper CI-limit No of obs.
0           1919 1956           1992        372
1           1891 2042           2192         18

# chickwts-data
data(chickwts)
myGroupedTest(chickwts[, 1], chickwts[, 2], 0.05)

          Lower CI-limit  Mean Upper CI-limit No of obs.
casein             282.6 323.6          364.5         12
horsebean          132.6 160.2          187.8         10
linseed            185.6 218.8          251.9         12
meatmeal           233.3 276.9          320.5         11
soybean            215.2 246.4          277.7         14
sunflower          297.9 328.9          359.9         12
```

6 Summera slumpmässigt antal tärningar del 1 (5 p)

Funktionen `sumOfDice()` 14 i sektionen Slumptal och Statistik summerar värdet från ett fixt antal tärnigar, nu ska ni skriva en funktion som kan summera ett slumpmässigt antal tärningar. Träningar syftar här på D6 som de beskrivs i 14. Skapa en ny funkton `sumOfRandomDice(K=,lambda=,mySeed=)`. Argumenten ska vara:

- K= antal dragningar från slumpfördelningen
- lambda= ett positivt kontinuerligt tal
- mySeed=en seed som styr slumptalsgenereringen, default ska vara NULL.

Funktionen `sumOfRandomDice()` ska göra följande:

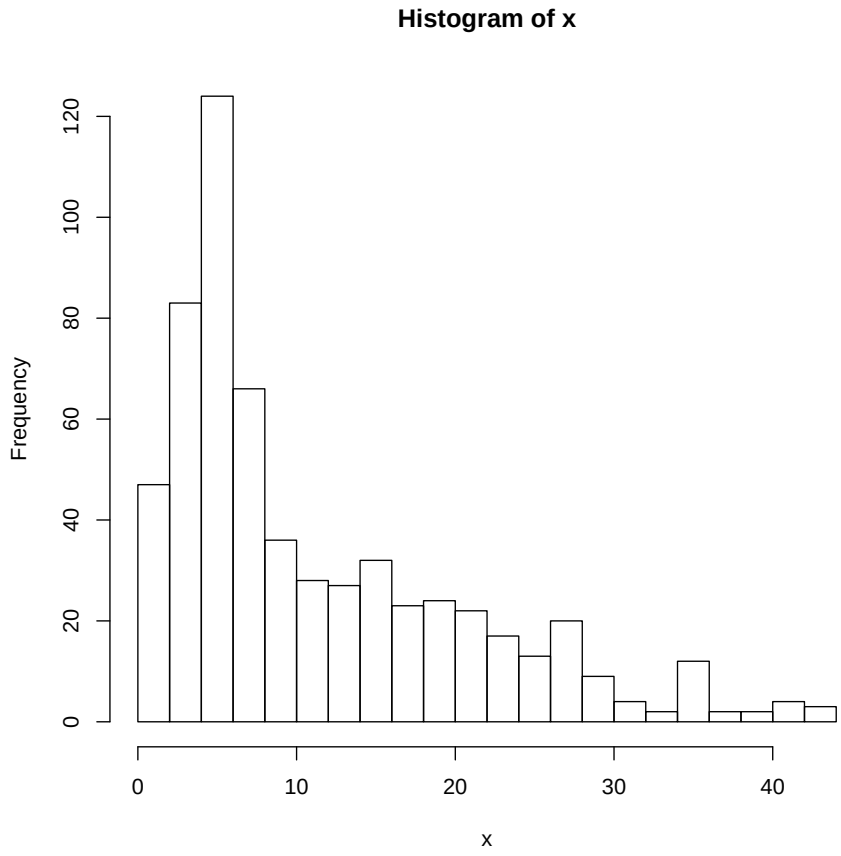
- Ändra seeden till: `set.seed(mySeed)`.
- Sätt upp en tom matris `result`, som ska ha `K` rader och 2 kolumner. Första komunen ska ha namnet `Value` och den andra `Dice`.
- Gör följande for-loop över vektorn `1:K`
  - Dra ett slumptal från en poissionfördelning med parameter=`lambda`. Spara slumptalet i `curretNumber`, vilket är antalet tärningar. Spara `curretNumber` i kolumnen `Dice` på aktuell rad i `result`.
  - Anropa `sumOfDice()` med argumenten K=1 och N=currentNumber, spara resultatet i kolumnen `Value` på aktuell rad i `result`. Observera att om `curretNumber=0` så ska summan bli `0`.
- Returnera `result`.

Sammanfattningsvis: i varje iteration ska ni dra ett slumptal (`currentNumber`), som anger antalet trärningar, sen ska ni summera `currentNumber` tärnigar. Detta upprepas så att ni får `K` stycken slumptal. Tex, första iterationen: vi drar talet 4 från en fördelningen $Poission(lambda = 3)$, sen kastar vi 4 tärningar och summerar dessa. Andra iterationen: vi drar talet 8 från en fördelningen $Poission(lambda = 3)$, sen kastar vi 8 tärningar och summerar dessa. Vi upprepar tills vi har `K` stycken tal.

Testa om testfallen nedan fungerar:

```
# test 1:
sumOfRandomDice(K = 5, lambda = 3, mySeed = 123)

     Value Dice
[1,]     8    2
[2,]    21    5
```

```
[3,]    14    3
[4,]     9    3
[5,]     0    0

# test 2:
sumOfRandomDice(K = 5, lambda = 8, mySeed = 543)

     Value Dice
[1,]    40   12
[2,]    32    8
[3,]    21    6
[4,]    49   16
[5,]    34    8

# test 3:
x <- sumOfRandomDice(K = 300, lambda = 5, mySeed = 387)
hist(x, 20)
```




```
# test 4:
y <- sumOfRandomDice(K = 100, lambda = 10, mySeed = 723)
mean(y)

[1] 22.38

sd(y)

[1] 15.61

# test 5:
z <- sumOfRandomDice(K = 30, lambda = 0.4, mySeed = 395)
table(z[, 1], z[, 2])


     0  1  2  3  4
  0 19  0  0  0  0
```

```
 1   0  4  0  0  0
 2   0  2  0  0  0
 6   0  2  0  0  0
 9   0  0  1  0  0
 10  0  0  0  1  0
 20  0  0  0  0  1
```

7 Tid för blodgivning (5 p)

För blodgivare finns vissa regler för när hen får ge blod. Här står det vilka regler som gäller. Ni ska skriva en funktion som ska hjälpa en blodgivare att veta när den får ge blod, utifrån några av reglerna. Funktionen ska heta `giveBlood()` och ha argumenten:

- `lasttime`: ett datum som anger senaste gången bolodgivaren gav blod, default ska vara idag. tips: today()
- `holiday`: ska vara antingen: 1) ett interval-objekt som anger start- och slutdatum för en utlandsresa. Startdatum är det datum som personen lämnar Sverige och slutdatum är det datum som personen kommer hem till Sverige. 2) Defaultvärde ska vara "hemma", vilket indikerar att det inte blir någon resa.
- `sex`: antar värdet "f" för kvinna och "m" för man
- `typeOfTravel`: "malaria" indikerar resa till ett land där det finns malaria och "other" indikerar resa till ett land utan malaria. Ska vara NULL om `holiday` har värdet "hemma".

Alla datum ska vara på formen **"year-month-day"**. Funktionen ska givet argumenten räkna ut ett datum när blodgivaren får ge blod igen och returnera datumet. Vi utgår ifrån att blodgivaren vill ge blod så **ofta** som möjligt. Funktionen ska följa följande regler:

- Minsta tid mellan blodgivningstillfällen: kvinnor 4 månader, män 3 månader, båda anger relativ tid.
- Om personen varit i ett land där det inte finns malaria ska den vänta 4 veckor (relativ tid) efter slutdatum i argumentet `holiday` innan den får ge blod.
- Om personen varit i ett land där det finns malaria ska den vänta 6 månader (relativ tid) efter slutdatum i argumentet `holiday` innan den får ge blod.
- Vi utgår ifrån att blodgivningscentralen bara är öppen på vardagar (måndag till fredag), så givet de tidigare reglerna så ska den första möjliga vardagen väljas.

Nedan följer ett förslag på lösningsordning:

1. Undersök om personen varit hemma, på resa i land med malaria eller i land utan malaria. Addera eventuell tilläggstid till slutdatum och spara som `extraTime`. Tänk på att ta hänsym till fallet då personen inte reser, tex genom att sätta `extraTime` till samma datum som `lasttime`. Tips: `int_end(), months(), weeks()`

2. Givet om den är en man eller kvinna räkna ut när personen tidigast får ge blod, spara det datumet i variablen `suggestion`. Tips: `months()`

3. Kolla om `suggestion` inträffar efter `extraTime`, om så är fallet ange `suggestion` som förslag för blodgivnig. Om så inte är fallet, ange dagen efter `extraTime` som förslag.

4. Kontrollera att den angivna dagen är en vardag, om inte ange nästa vardag som förslag. Tips: `?weekday(), ?days()`

5. Returnera förslaget som en text-sträng på formen:
   "year=[året], month=[månaden], day=[dagen], weekday=[veckodagen]".
   Tex om föreslaget datum är 2014-02-21 så ska strängen bli:
   "year=2014, month=Feb, day=19, weekday=Friday".
   Tips: `year(), month(), day(), paste()`

Kolla om funktionen uppfyller testfallen nedan:

```
# test 1:
day1 <- ymd("2014-02-24")
giveBlood(lasttime = day1, holiday = "hemma", sex = "m", typeOfTravel = NULL)

[1] "year=2014 month=May day=26 weekday=Monday"

giveBlood(lasttime = day1, holiday = "hemma", sex = "f", typeOfTravel = NULL)

[1] "year=2014 month=Jun day=24 weekday=Tuesday"

# test 2:
day2 <- ymd("2014-03-23")
day3 <- ymd("2014-04-24")
holiday1 <- new_interval(day2, day3)
giveBlood(lasttime = day1, holiday = holiday1, sex = "m", typeOfTravel = "malaria")

[1] "year=2014 month=Oct day=27 weekday=Monday"

giveBlood(lasttime = day1, holiday = holiday1, sex = "f", typeOfTravel = "malaria")

[1] "year=2014 month=Oct day=27 weekday=Monday"
```

```r
# test 3:
day4 <- ymd("2014-04-13")
day5 <- ymd("2014-05-23")
holiday2 <- new_interval(day4, day5)
giveBlood(lasttime = day1, holiday = holiday2, sex = "m", typeOfTravel = "other")
```

```
[1] "year=2014 month=Jun day=23 weekday=Monday"
```

```r
giveBlood(lasttime = day1, holiday = holiday2, sex = "f", typeOfTravel = "other")
```

```
[1] "year=2014 month=Jun day=24 weekday=Tuesday"
```

Nu är du klar!