Part I

Exercises

1 Introduction to R

Ex. 1 — Download and install R from the CRAN http://cran.r-project.org/. Bonus points if you compile it yourself.

Ex. 2 — Download and install Rstudio from http://www.rstudio.com/.

Ex. 3 — Install the **devtools** package from the CRAN using R's `install.packages()` function.

Ex. 4 — Create an R script that contains the code

```
print("Hello, world!")
```

Run this script from an interactive R session using the `source()` command, and run this script in batch from the shell using `Rscript`.

Ex. 5 — In each case, what is the value of `x`? (Try to think it through before you try it in R)

```
x <- 2 - 1 * 2
x <- 6/3-2+1*0+3/3-3
x <- 19%%17%%13 # compare to (19%%17)%%13 and 19%%(17%%13)
x <- 2^17%%17
x <- 3-2%%5+3*2-4/2
```

Ex. 6 — In each case, what is the value of `x`?

```
x <- sum(1:10-5)
x <- sum(1:10)-5
x <- 1:10-10:1
x <- sum(1:10-10:1)
```

Ex. 7 — Russian Roulette is a game played with a revolver and a single bullet. Write an R script which simulates the game of Russian Roulette.

2 Data Structures

Ex. 8 — Give the R code required to produce this list:

```
$a
[1] 1 2 3 4 5

$b
[1] "a" "b"

$c
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Ex. 9 — Take the vector

```
set.seed(1234)
x <- c(rep(NA, 3), rnorm(5))
```

and use the `sum()` function on this vector. Experiment with its options for handling `NA`'s (see `?sum` for details).

Ex. 10 — The sample mean of a vector $x = [x_i]_{i=1}^n$ is defined as

$$mn_x = \sum_{i=1}^{n} \frac{x_i}{n}$$

and the unbiased sample variance is defined as

$$var_x = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - mn_x)^2$$

Write an R script which will compute the mean and variance of the vector `x <- 1:100`. Compare with R's internal `mean()` and `var()` functions.

Ex. 11 — R has a full suite of matrix-vector and matrix-matrix arithmetic operations, including `t()`, `+`, `*`, and `%*%`. Let

```
x <- matrix(1:30, 10)
y <- matrix(1:10, 2)
a <- 1:3
b <- 1:5
```

and experiment with these different operations, such as `x+a`, `x*x`, `t(x) %*% x`, etc.

Ex. 12 — A square matrix A is said to be invertible if there exists a matrix B such that

$$AB = BA = I$$

Where I is the identity matrix (1 along the diagonal, 0 elsewhere). Use R to compute the inverse of this matrix:

```r
set.seed(1234)
x <- matrix(rnorm(25), nrow=5, ncol=5)
```

3 Control Flow

Ex. 13 — In each case, what is the value of `x`?

```
x <- 1==1
x <- 1==0
x <- !1==1
x <- if (1==0) 0 else 1
x <- ifelse(1==0, 1, 0)
x <- if (1==TRUE) F else T
x <- if (10) 1 else 0
x <- if (1==0) 0
```

Ex. 14 — In each case, what is the value of `x`?

```
x <- 1==0 || 1>0
x <- 1==0 && 1 > 0
x <- !(1==0) || !(1>0)
x <- (!1==0 && !1>0) == !(1==0 || 1>0)
x <- if (x <- 10 > 0 && TRUE < 11) TRUE else FALSE
```

Ex. 15 — An *infinite loop* is a bug where your program is trapped inside of a loop forever. The simplest infinite loop is

```
while(TRUE){}
```

So that we can see what's really going on, here is a more instructive example:

```
i <- 0
while(TRUE){
  i <- i+1
  print(i)
}
```

Try running this code, and remember that the key sequence `ctrl+c` breaks computation (including infinite loops).

Ex. 16 — Suppose we wish to take the square root of the numbers 1 to 1000. Do this in each of the following ways, timing each one:

- for loop without initialization
- for loop with initialization
- lapply
- vectorized

Ex. 17 — Project Euler is a website that contains hundreds of programming challenges, and is an invaluable resource for learning a new language (or programming in general). Go to `https://projecteuler.net`, register an account, and solve the first problem using R: `https:`

//projecteuler.net/problem=1. Make sure that your program solves the problem in under a second.

Ex. 18 — "Fizzbuzz" is a simple programming challenge often used at interviews to test very basic programming skill. Your goal is the following: for the numbers 1 to 100, print "fizz" if the number is a multiple of 3, "buzz" if the number is a multiple of 5, "fizzbuzz" if the number is a multiple of *both* 3 and 5, and simply print the number otherwise.

Ex. 19 — The Fibonacci Numbers are an important integer sequence. The sequence is usually recursively defined as follows:

- $F_0 = F_1 = 1$
- For $n > 1$, $F_n = F_{n-1} + F_{n-2}$

So for example $F_2 = F_1 + F_0 = 1 + 1 = 2$. Write an R script that, for any non-negative integer n produces the n'th Fibonacci Number. Test that your script works by verifying that $F_{25} = 121393$.

Ex. 20 — Find all primes smaller than 1000.

Ex. 21 — Newton's Method is an algorithm for finding approximations to roots (values where a function is 0). It says that if f is a real-valued function of a real variable, with derivative f'. Take initial guess x_0, and for each $n > 0$, define

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$$

Then it can be shown that under certain conditions, the sequence of x's will converge to a root of f.

Let $f(x) = x^4 - 9x^3 - 334x^2 + 4416x - 10080$, so that $f'(x) = 4x^3 - 27x^2 - 668x + 4416$. Take an initial, uniform random guess from -10 to 10 and perform 50 iterations of Newton's Method to find an approximation of a root. This polynomial has 4 real roots; which did you find?

Ex. 22 — Imagine a high school with 1000 lockers all in a row, numbered 1 to 1000 in order. At the start, all of them are closed. 1000 students are sent, one after the other, to change the state of a set of lockers (from open to closed or closed to open). The first student changes the state of all lockers. The second changes the state of every other one (2, 4, 6, 8, ...). The third changes the state of every third one (3, 6, 9, 12, ...). This process continues until all 1000 students have gone. Which lockers are open at the end of this process? For bonus points, explain *why* these are the remaining lockers.

4 Functions

Ex. 23 — Write a function that takes 3 numbers a, b, and c as inputs and returns the smallest number of the three.

Ex. 24 — Write a function that takes an array of numbers x and returns the smallest number in the array.

Ex. 25 — Dungeons and Dragons is a game played with polyhedral dice beyond the standard 6-sided dice. For example, you might need to roll a 20-sided die 3 times and sum the result; this is referred to as "3d20" as a kind of shorthand. Write a function which will mdn for integers m and n.

Ex. 26 — Take the code from Exercise 18 and turn it into a function, where instead of stopping at 100, the printing will stop at a user-supplied input `n`.

Ex. 27 — Take the code from Exercise 19 and turn it into a function that produces the n'th Fibonacci number.

Ex. 28 — Take the code from Exercise 22 and turn it into a function that produces a vector of the open lockers, starting from an initial set of `n` lockers, where `n` is a user-supplied input.

Ex. 29 — The factorial of a non-negative integer n, notated $n!$, can be algebraically defined as

$$\begin{aligned} n! &= \prod_{i=0}^{n-1} (n-i) \\ &= n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 2 \cdot 1 \end{aligned}$$

In the case where $n = 0$, $n! = 1$ (empty products are always 1). Write a function which recursively computes the factorial. Write a second function which computes the factorial without recursion.

Ex. 30 — Write a function which recursively computes the n'th Fibonacci number.

Ex. 31 — Zeller's congruence is an algorithm for taking a date and returning the day of the week (Monday, Tuesday, ...). It says that the day of the week h is given by:

$$h = \left(q + \left\lfloor \frac{13(m+1)}{5} \right\rfloor + K + \left\lfloor \frac{K}{4} \right\rfloor + \left\lfloor \frac{J}{4} \right\rfloor - 2J \right) \mod 7$$

Where h is the index of the day of the week (0 for Saturday, 1 for Sunday, ...), q is the day of the month, m is an indexing of the month (3 for March, 4 for April, ..., 14 for February — so January and February are counted with the actual previous year), K is the year $\mod 100$, and J is $\left\lfloor \frac{year}{100} \right\rfloor$. Write a function which takes in the day, month and year (by ordinary accounting), and returns the day of the week. Test that your function is working with the fact that January 1, 2013 was on a Tuesday, and March 1, 2013 was on a Friday.

5 Strings

Ex. 32 — Figure out what this string of gibberish is actually doing:

```
x <- rnorm(1000, mean=10)

cat(sprintf("mean:\t%.2f\nvar:\t%.2f\n", mean(x), var(x)))
```

Ex. 33 — Use paste() on these character vectors:

```
a <- c("o", "a", "a", "n", "Y", "f", "d", "e", "c", " ", "s", ".")
b <- c("g", "u", "i", "!", "u", "u", "t", "s", "e", "e", "g", "")
c <- c("n", "t", "t", "s", "o", "o", " ", " ", "r", "m", "a", "")
d <- c("C", "r", "l", "o", " ", " ", "n", "h", "e", "t", "s", "e")
```

to decode the secret message.

Ex. 34 — Write a function to reverse the case of a string (all caps are sent to lowercase and all lowercase are capitalized).

Ex. 35 — A *palindrome* is a string which is the same even after it is reversed, such as “12321” or “racecar”. Determine whether or not the following two numbers are palindromes:

- 7313251416848880330652670035390409218376812461217106834138522783887608393519025808100767202597280997030718964589854698170307990827952027670018085209153938067883872258314386017121642186738129040935300762560330888486141523137

- 7313251416848880330652670035390409218376812461217106834138522783887608393519025808100767202597280997030718964589854698170307990827952027670078085209153938067883872258314386017121642186738129040935300762560330888486141523137

Ex. 36 — Write a function `chomp()` that, given a string, removes from the string any occurrence of the character `&`, as well as the character to the left of each `&` character. So for example, your function should return:

```
> chomp("a&c")
[1] "c"
> chomp("a&")
[1] ""
> chomp("abc")
[1] "abc"
```

6 IO

Throughout, let `x <- data.frame(a=1:5, b=5:1)`.

Ex. 37 — Save `x` to disk using `write.csv()` and `write.table()`. Experiment with the different options.

Ex. 38 — Use `read.csv()` and `read.table()` to read the file(s) you wrote in Exercise 37.

Ex. 39 — Use `scan()` to read the file(s) you wrote in Exercise 37.

Ex. 40 — Repeat Exercises 37 and 38 using R's `save()` and `load()` functions.

7 Plotting

Ex. 41 — Create a scatterplot of two random normal variables.

Ex. 42 — Save the plot from Exercise 41 as a png and as a pdf.

Ex. 43 — Install the **ggplot2** package from the CRAN using `install.packages()`.

8 Debugging

Ex. 44 — Find the bug:

```
x <- 0:9
if (x[1] = 1){
  print(x)
}
```

Ex. 45 — Find the bug:

```
x <- 0:9
if (x[0] == 1){
  print(x)
}
```

Ex. 46 — Find the bug:

```
myfactorial <- function (x)
{
  if (x==1)
    return(1)
  else
    return( x*myfactorial(x) )
}
```

Ex. 47 — Use the `debug()` function to debug this function:

```
f <- function(X)
{
  scl <- sum(as.numeric(X$a))
  ans <- scl * (as.numeric(X$a)+X$b)
  ans <- crossprod(ans)

  return(ans)
}

X <- list(a=factor(-2:2), b=matrix(1:30, nrow=10))
f(X)
```

So that for input

```
X <- matrix(1:30, 10)
```

it produces the correct output:

```
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
[3,]    0    0    0
```

Ex. 48 — Find the bug:

```
f <- function(n)
{
  if (n==1)
    return(1)
  else {
    if (n%%2==0)
      return(n/2)
    else
      return(3*x)
  }
}

x <- 1
f(x)
n <- 3
f(n)
```

Part II

Hints and Solutions

Answer (Ex. 5) — The values of `x` are:

```
[1] 0
[1] -2
[1] 2
[1] 2
[1] 5
```

Answer (Ex. 6) — The values of `x` are:

```
[1] 5
[1] 50
[1] -9 -7 -5 -3 -1  1  3  5  7  9
[1] 0
```

Answer (Ex. 7) — One possible solution is:

```
sample(c("BANG", "click"), size=1, prob=c(1/6, 5/6))
```

Answer (Ex. 10) — A possible solution is:

```
x <- 1:100
mn <- sum(x)/length(x)
var <- sum((x-mn)^2)/(length(x)-1)
```

Answer (Ex. 13) — The values of `x` are:

```
[1] TRUE
[1] FALSE
[1] FALSE
[1] 1
[1] 0
[1] FALSE
[1] 1
[1] NULL
```

Answer (Ex. 14) — The values of `x` are:

```
[1] TRUE
[1] FALSE
[1] TRUE
[1] TRUE
[1] TRUE
```

Answer (Ex. 16) — See the lecture notes.

Answer (Ex. 18) — One possible solution is:

```
for (i in 1:100){
  if (i%%3==0) {
    if (i%%5==0){
      print("fizzbuzz")
    } else {
      print("fizz")
    }
  } else if (i%%5==0) {
    print("buzz")
  } else {
    print(i)
  }
}
```

Answer (Ex. 19) — One possible solution is:

```
n <- 26

if (n==0 || n==1){
  print(1)
} else {
  prev1 <- prev2 <- 1
  i <- 2
  while(i < n){
    fib <- prev1 + prev2
    prev2 <- prev1
    prev1 <- fib
    i <- i+1
  }
}

print(fib)
```

Answer (Ex. 20) — Using the Sieve of Eratosthenes:

```
x <- 2:1000

n <- sqrt(max(x))

for (i in x[1:n]){
  for (j in x[which(x>i)]){
    if (j%%i==0){
      x[j-1] <- 0
    }
  }
}

x[which(x>0)]
```

Answer (Ex. 21) — You probably found 3:

```
x <- runif(1, -10, 10)

i <- 1
while(i<20){
  x <- x -
     (x^4-9*x^3-334*x^2+4416*x-10080)/(4*x^3-27*x^2-668*x+4416)
  i <- i+1
}

x
```

It would really be better to take line 5 above and turn the two evaluations (at the function and the derivative) and turn them into R functions.

Answer (Ex. 22) — One possible solution is:

```
lockers <- rep(0, 1000)

for (i in 1:1000){
  for (j in seq(from=i, to=1000, by=i)){
    lockers[j] <- 1-lockers[j]
  }
}

which(lockers>0)
```

Answer (Ex. 23) — One possible solution is:

```
f <- function(a, b, c)
```

```
{
  ans <- min(a, b, c)
  return(ans)
}
```

Another more instructive one is:

```
g <- function(a, b, c)
{
  ans <- a
  if (b < ans){
    ans <- b
  }

  if (c < ans){
    ans <- c
  }

  return(ans)
}
```

Answer (Ex. 24) — One possible solution is:

```
f <- function(x)
{
  ans <- min(x)
  return(ans)
}
```

Another more instructive one is:

```
g <- function(x)
{
  ans <- x[1]
  for (i in 2:length(x)){
    if (x[i] < ans){
      ans <- x[i]
    }
  }

  return(ans)
}
```

Answer (Ex. 25) — One possible solution is:

```
dice <- function(m, n)
{
  ans <- sum(sample(1:n), size=m, replace=TRUE)
```

```
  return(ans)
}
```

Answer (Ex. 29) — For the first:

```
factorial1 <- function(n)
{
  if (n <= 2){
    return(n)
  } else {
    return(n*factorial1(n-1))
  }
}
```

And for the second:

```
factorial2 <- function(n)
{
  if (n <= 2){
    ans <- n
  } else {
    ans <- 2
    while(n > 2){
      ans <- ans*n
      n <- n-1
    }
  }

  return(ans)
}
```

Answer (Ex. 30) — One possible solution is:

```
fibonacci <- function(n)
{
  if (n==0 || n==1) {
    return(n)
  } else {
    return( fibonacci(n-1) + fibonacci(n-2) )
  }
}
```

Answer (Ex. 32) — `cat()` "concatenates representations"; sort of like function evaluation for strings. `sprintf()` is the **s**tring **f**ormatted **print**. `\n` is a newline character and `\t` is a tab. Unpacking the rest should be simple.

Answer (Ex. 33) — The answer is `Congratulations! You found the secret message.`, which you can see for yourself by entering:

```
paste(d, a, c, b, collapse="", sep="")
```

Answer (Ex. 34) — One possible solution is:

```
swap_case_char <- function(char)
{
  if (char %in% letters)
    return( LETTERS[which(letters==char)] )
  else if (char %in% LETTERS)
    return( letters[which(LETTERS==char)] )
  else
    return(char)
}

swap_case <- function(s)
{
  if (!is.character(s))
    stop("Input 's' must be of character type.")

  ltrs <- unlist(strsplit(s, split=""))
  ltrs <- sapply(ltrs, swap_case_char)
  swapped <- paste(ltrs, collapse="")

  return(swapped)
}
```

Answer (Ex. 35) — Yes and no, respectively:

```
x <-
"731325141684888033065267003539040921837681246121710683413852278388760839
808100767202597280997030718964589854698170307990827952027670018085209153
387225831438601712164218673812904093530076256033088486141523137"

y <-
"731325141684888033065267003539040921837681246121710683413852278388760839
808100767202597280997030718964589854698170307990827952027670078085209153
387225831438601712164218673812904093530076256033088486141523137"

is.palindrome <- function(str)
{
  paste(rev(unlist(strsplit(str, split=""))), collapse="") == str
}

is.palindrome(x)
is.palindrome(y)
```

Answer (Ex. 36) — One possible solution is:

```
chomp <- function(s)
{
  ltrs <- unlist(strsplit(s, split=""))

  stars <- which(ltrs=="&")

  leftofs <- stars-1
  leftofs <- leftofs[which(leftofs>0)]

  removes <- c(leftofs, stars)

  if (length(removes) > 0)
    ltrs <- ltrs[-removes]

  ret <- paste(ltrs, collapse="")

  return(ret)
}
```

Answer (Ex. 44) — Use `==` for comparison, not `=` (which can be used for assignment).

Answer (Ex. 45) — Vectors in R are indexed from 1, not 0 (like in C). The vector `x` contains no 0'th element.

Answer (Ex. 46) — Calling `f(x)` from inside any function `f` will cause infinite recursion. The call should instead be `x*myfactorial(x-1)`.

Answer (Ex. 47) — Remember that conversion of factors to numeric data is often not straightforward. Try casting the factor as character first in the `scl <-` assignment.

Answer (Ex. 48) — Type `rm(x)` then re-run `f(n)`. Now look at your variable names in the function definition...