Essentials

Block 1, Exercises

Exercises for the first block, Essentials.

Exercises, Chapter 2

Exercise 2.1

```
100 * ( 1 + (0.05 / 12) )^24
```

```
## [1] 110.4941
```

Exercise 2.2

```
5 %% 2
```

```
## [1] 1
```

Exercise 2.3

```
3333 %% 222
```

```
## [1] 3
```

Exercise 2.4

```
domainValues<-10^(c(1:10))
# avoid scientific notation
options(scipen=1000)
# increase significant digits
options(digits=22)
# applying formula
rangeValues<- ( 1 + 1/domainValues)^domainVal
# force output to a single column of values
options(width=40)
rangeValues
```

```
##  [1] 2.593742460100002311663
##  [2] 2.704813829421528481589
##  [3] 2.716923932235593586171
##  [4] 2.718145926824925506793
##  [5] 2.718268237192297487326
##  [6] 2.718280469095753382192
##  [7] 2.718281694132081760529
##  [8] 2.718281798347357725021
##  [9] 2.718282052011560256943
## [10] 2.718282053234787554175
```

Exercises, Chapter 3

Exercise 3.1

```
D<-1000
K<-5
h<-0.25
# implementing square root as an exponent: raised to 1/2
Q<-((2*D*K)/h)^(0.5)
Q
```

```
## [1] 200
```

```
ls()
```

```
## [1] "D"            "domainValues"
## [3] "h"            "K"
## [5] "Q"            "rangeValues"
```

Exercise 3.2

```
P<-100; r<-0.08; n<-12; t<-3
F<-P*(1+r/n)^(n*t)
F
```

```
## [1] 127.0237051620650703398
```

Exercises, Chapter 4

Exercise 4.1

```
desiredVector<-rep(c(2.7,8,3),2)
desiredVector
```

```
## [1] 2.700000000000000177636
## [2] 8.000000000000000000000
## [3] 3.000000000000000000000
## [4] 2.700000000000000177636
## [5] 8.000000000000000000000
## [6] 3.000000000000000000000
```

Exercise 4.2

```
desiredVector<-seq(0,2,by=.4)
desiredVector
```

```
## [1] 0.0000000000000000000000
## [2] 0.4000000000000000222045
## [3] 0.8000000000000000444089
## [4] 1.2000000000000001776357
## [5] 1.6000000000000000888178
## [6] 2.0000000000000000000000
```

Exercise 4.3

```
primes      <-  c(2,3,5,7,11,13,17,19,23,29)
composites  <-  c(4,6,8,9,10)
primes[composites]
```

```
## [1]  7 13 19 23 29
```

Exercise 4.4

```
seq(3, 28, by=11) %/% 4
```

```
## [1] 0 3 6
```

Exercise 4.5

```
seq(0, 2, length.out=5)
```

```
## [1] 0.0 0.5 1.0 1.5 2.0
```

Exercise 4.6

```
x<-c(2,0,-5,-7)
x
```

```
## [1]  2  0 -5 -7
```

```
# negative index means show all in x except for index
# here -2.8, which is truncated to -2,
# therefore, show all of x except for the second element
x[-2.8]
```

```
## [1]  2 -5 -7
```

Exercise 4.7

```
rep(0:2,1:3)
```

```
## [1] 0 1 1 2 2 2
```

```
3 ^ rep(0:2,1:3)
```

```
## [1] 1 3 3 9 9 9
```

Exercise 4.8

```
seed <- rep(1:4)
desiredVector <- c(seed,1+seed,2+seed,3+seed)
desiredVector
```

```
##  [1] 1 2 3 4 2 3 4 5 3 4 5 6 4 5 6 7
```

```
# also
rep(1:4,4)+c(rep(0,4),rep(1,4),rep(2,4),rep(3,4))
```

```
##  [1] 1 2 3 4 2 3 4 5 3 4 5 6 4 5 6 7
```

Exercise 4.9

```
# end points fixed at 0/2 and 8/2
# increment of 0.05 = 0.1/2
seq(0,8,by=.1)/2
```

```
##  [1] 0.0000000000000000000000
##  [2] 0.0500000000000000027756
##  [3] 0.1000000000000000055511
##  [4] 0.1500000000000000222045
##  [5] 0.2000000000000000111022
##  [6] 0.2500000000000000000000
##  [7] 0.3000000000000000444089
##  [8] 0.3500000000000000333067
##  [9] 0.4000000000000000222045
## [10] 0.4500000000000000111022
## [11] 0.5000000000000000000000
## [12] 0.5500000000000000444089
## [13] 0.6000000000000000888178
## [14] 0.6500000000000000222045
## [15] 0.7000000000000000666134
## [16] 0.7500000000000000000000
## [17] 0.8000000000000000444089
## [18] 0.8500000000000000888178
## [19] 0.9000000000000000222045
## [20] 0.9500000000000000666134
## [21] 1.0000000000000000000000
## [22] 1.0500000000000000444089
## [23] 1.1000000000000000888178
## [24] 1.1500000000000001332268
## [25] 1.2000000000000001776357
## [26] 1.2500000000000000000000
## [27] 1.3000000000000000444089
## [28] 1.3500000000000000888178
## [29] 1.4000000000000001332268
## [30] 1.4500000000000001776357
## [31] 1.5000000000000000000000
## [32] 1.5500000000000000444089
## [33] 1.6000000000000000888178
## [34] 1.6500000000000001332268
## [35] 1.70000000000000017763568
## [36] 1.75000000000000000000000
## [37] 1.80000000000000004440892
## [38] 1.85000000000000008881784
## [39] 1.90000000000000013322676
## [40] 1.95000000000000017763568
## [41] 2.00000000000000000000000
## [42] 2.05000000000000026645353
## [43] 2.10000000000000008881784
## [44] 2.14999999999999991118216
## [45] 2.20000000000000017763568
## [46] 2.25000000000000000000000
## [47] 2.30000000000000026645353
## [48] 2.35000000000000008881784
## [49] 2.40000000000000035527137
## [50] 2.45000000000000017763568
## [51] 2.50000000000000000000000
## [52] 2.55000000000000026645353
## [53] 2.60000000000000008881784
## [54] 2.65000000000000035527137
## [55] 2.70000000000000017763568
## [56] 2.75000000000000000000000
## [57] 2.80000000000000026645353
## [58] 2.85000000000000008881784
## [59] 2.90000000000000035527137
## [60] 2.95000000000000017763568
## [61] 3.00000000000000000000000
## [62] 3.05000000000000026645353
## [63] 3.10000000000000008881784
## [64] 3.15000000000000035527137
## [65] 3.20000000000000017763568
## [66] 3.25000000000000000000000
## [67] 3.30000000000000026645353
## [68] 3.35000000000000008881784
## [69] 3.40000000000000035527137
## [70] 3.45000000000000017763568
## [71] 3.50000000000000000000000
## [72] 3.55000000000000026645353
## [73] 3.60000000000000008881784
## [74] 3.65000000000000035527137
## [75] 3.70000000000000017763568
## [76] 3.75000000000000000000000
## [77] 3.80000000000000026645353
## [78] 3.85000000000000008881784
## [79] 3.90000000000000035527137
## [80] 3.95000000000000017763568
## [81] 4.00000000000000000000000
```

Exercise 4.10

```
x <- seq(1:8)
x
```

```
## [1] 1 2 3 4 5 6 7 8
```

```
x[6:8]
```

```
## [1] 6 7 8
```

```
x[c(6:8)]
```

```
## [1] 6 7 8
```

```
x[-c(-6:-8)]
```

```
## [1] 6 7 8
```

Exercise 4.11

```
(1:10)^(1:10)
```

```
##  [1]           1           4          27
##  [4]         256        3125       46656
##  [7]      823543    16777216   387420489
## [10] 10000000000
```

Exercises, Chapter 5

Exercise 5.1

```
rbind(5:9,rep(12,5),(9:5)^2,c(2,8,4,7,3))
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    5    6    7    8    9
## [2,]   12   12   12   12   12
## [3,]   81   64   49   36   25
## [4,]    2    8    4    7    3
```

Exercise 5.2

```
x<-matrix(11:18,2,4)
x
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   11   13   15   17
## [2,]   12   14   16   18
```

```
x[-1,c(1,4)]
```

```
## [1] 12 18
```

Exercise 5.3

```
A<-matrix(1:6,3,2)
B<-A+6
A
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

```
B
```

```
##      [,1] [,2]
## [1,]    7   10
## [2,]    8   11
## [3,]    9   12
```

```
rbind(A,B)
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
## [4,]    7   10
## [5,]    8   11
## [6,]    9   12
```

```
cbind(A,B)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

Exercise 5.4

```
# easier to read and infer
A<-c(0.8,0.3)
cbind(A,1-A)
```

```
##                             A
## [1,] 0.8000000000000000444089
## [2,] 0.2999999999999999888978
##
## [1,] 0.1999999999999999555911
## [2,] 0.6999999999999999555911
```

```
# or, single command; may expect values in ob
cbind(c(0.8,0.3),1-c(0.8,0.3))
```

```
##                          [,1]
## [1,] 0.8000000000000000444089
## [2,] 0.2999999999999999888978
##                          [,2]
## [1,] 0.1999999999999999555911
## [2,] 0.6999999999999999555911
```

Exercise 5.5

```
matrix(rep(1:4,1:4),2,5,T)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    2    3    3
## [2,]    3    4    4    4    4
```

Exercises, Chapter 6

Exercise 6.1

- text, p.32, ‘The `array` function is a generalization of the `matrix` function.’
- A *matrix* is an *array* whose third and final subscript has a value of one.
- In my notes, I have the phrase “single layer” written next to this question.

Exercise 6.2

```
# x<-array()        create array, x,
# seq(1:18)         populated with first 18 positive integ
# dim<-c(3,3,2)     with dimensions 3x3x2
#                   using default input convention
x<-array(seq(1:18),dim<-c(3,3,2))
x
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]   10   13   16
## [2,]   11   14   17
## [3,]   12   15   18
```

```
# x[]   extract elements of x
# 2:3   include second and third rows
# -2    exclude second column
# 2     include second layer
x[2:3,-2,2]
```

```
##      [,1] [,2]
## [1,]   11   17
## [2,]   12   18
```

Exercise 6.3

```
a<-array(seq(1:60),dim<-c(3,4,5))
a[c(1,3),-c(1,4),3:5]
```

```
## , , 1
##
##      [,1] [,2]
## [1,]   28   31
## [2,]   30   33
##
## , , 2
##
##      [,1] [,2]
## [1,]   40   43
## [2,]   42   45
##
## , , 3
##
##      [,1] [,2]
## [1,]   52   55
## [2,]   54   57
```

```
a[-2,2:3,3:5]
```

```
## , , 1
##
##      [,1] [,2]
## [1,]   28   31
## [2,]   30   33
##
## , , 2
##
##      [,1] [,2]
## [1,]   40   43
## [2,]   42   45
##
## , , 3
##
##      [,1] [,2]
## [1,]   52   55
## [2,]   54   57
```

Exercises Chapter 7

Exercise 7.1

Write a single R command that calculates:

$sin(e^4 + \sqrt{arccos(1/3)})$

```
sin(exp(4) + sqrt(acos(1/3)))
```

```
## [1] -0.7453312958787164932417
```

Exercise 7.2

```
x <- c(pi, 4/3, 7)
x
```

```
## [1] 3.141592653589793115998
## [2] 1.333333333333333259318
## [3] 7.000000000000000000000
```

```
round(x,2)
```

```
## [1] 3.140000000000000124345
## [2] 1.330000000000000071054
## [3] 7.000000000000000000000
```

Exercise 7.3

```
x <- (1:10)^2
x
```

```
##  [1]   1   4   9  16  25  36  49  64  81
## [10] 100
```

```
diff(range(x))
```

```
## [1] 99
```

Exercise 7.4

```
x <- 1:4
y <- 4:1
pmax(x,y)
```

```
## [1] 4 3 3 4
```

Exercise 7.5

```
x <- 1:4
cumsum(x)
```

```
## [1]  1  3  6 10
```

Exercise 7.6

```
x <- c(1,1,1,-1,-1,-1,NA)
y <- c(0,1,4, 0,-1,NA, 1)
x / sqrt(y)
```

```
## Warning in sqrt(y): NaNs produced
```

```
## [1]  Inf  1.0  0.5 -Inf  NaN   NA   NA
```

Exercise 7.7

```
x <- seq(1,7,by=3)
sum(x^2) / length(x)
```

```
## [1] 22
```

```
x^2
```

```
## [1]  1 16 49
```

```
sum(x^2)
```

```
## [1] 66
```

```
length(x)
```

```
## [1] 3
```

Exercise 7.8

```
sqrt(max(9:-3))
```

```
## [1] 3
```

Exercise 7.9

```
options(width=25)
x<-seq(1,700,by=28)
x
```

```
##  [1]   1  29  57  85 113
##  [6] 141 169 197 225 253
## [11] 281 309 337 365 393
## [16] 421 449 477 505 533
## [21] 561 589 617 645 673
```

```
min(x)
```

```
## [1] 1
```

```
max(x)
```

```
## [1] 673
```

```
x<max(x)
```

```
##  [1]  TRUE  TRUE  TRUE
##  [4]  TRUE  TRUE  TRUE
##  [7]  TRUE  TRUE  TRUE
## [10]  TRUE  TRUE  TRUE
## [13]  TRUE  TRUE  TRUE
## [16]  TRUE  TRUE  TRUE
## [19]  TRUE  TRUE  TRUE
## [22]  TRUE  TRUE  TRUE
## [25] FALSE
```

```
x[x<max(x)]
```

```
##  [1]   1  29  57  85 113
##  [6] 141 169 197 225 253
## [11] 281 309 337 365 393
## [16] 421 449 477 505 533
## [21] 561 589 617 645
```

```
x[x<max(x)]>min(x)
```

```
##  [1] FALSE  TRUE  TRUE
##  [4]  TRUE  TRUE  TRUE
##  [7]  TRUE  TRUE  TRUE
## [10]  TRUE  TRUE  TRUE
## [13]  TRUE  TRUE  TRUE
## [16]  TRUE  TRUE  TRUE
## [19]  TRUE  TRUE  TRUE
## [22]  TRUE  TRUE  TRUE
```

```
x[x[x<max(x)]>min(x)]
```

```
##  [1]  29  57  85 113 141
##  [6] 169 197 225 253 281
## [11] 309 337 365 393 421
## [16] 449 477 505 533 561
## [21] 589 617 645
```

```
mean(x[x[x<max(x)]>min(x)])
```

```
## [1] 337
```

```
options(width=40)
```

Exercise 7.10

```
x <- matrix(1:6,2,3,T)
x
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

```
dim(x) <- c(3,2)
x
```

```
##      [,1] [,2]
## [1,]    1    5
## [2,]    4    3
## [3,]    2    6
```

Exercise 7.11

Write two R commands that calculate

$$\sum_{n=1}^{15}\left(\frac{2^i}{i!}-\frac{cos(3i)}{i^4}\right)$$

```
x<-1:15
sum( ( (2^x)/factorial(x) ) - ( cos(3*x) / (x^4) ) )
```

```
## [1] 7.327847608102410426056
```

Exercise 7.12

Write two R commands that calculate

$$\prod_{x=4}^{12} \left| \frac{x(x-1)(x-2)}{(x-3)!} + \frac{arctanx}{x^2} \right|$$

```
x <- 4:12
prod (
    abs ( ( x*(x-1)*(x-2) / factorial(x-3)  )
        + ( atan(x) / x^2 ) )
    )
```

```
## [1] 20.89520942794157676303
```

Exercise 7.13

Write a single R command that calculates:

$$\frac{3}{4} + \left(\frac{3}{4} \cdot \frac{5}{6}\right)\left(\frac{3}{4} \cdot \frac{5}{6} \cdot \frac{7}{8}\right) + \cdots + \left(\frac{3}{4} \cdot \frac{5}{6} \cdot \frac{7}{8} \cdots \frac{49}{50}\right)$$

```
sum(cumprod(seq(4,50,by=2)^-1*-1+1))
```

```
## [1] 8.452067611240138944595
```

Exercise 7.14

```
exp(exp(1))
```

```
## [1] 15.15426224147926224852
```

Exercise 7.15

Write a single R command that calculates:

$$1^3 + 2^3 + \cdots + 100^3$$

```
sum((1:100)^3)
```

```
## [1] 25502500
```

Exercise 7.16

Using a minimum number of keystrokes, write a single R command that creates a vector with elements:

$$\left(5, \frac{5^2}{2!}, \frac{5^3}{3!}, \ldots, \frac{5^{10}}{10!}\right)$$

```
5^(1:10)/factorial(1:10)
```

```
##  [1]  5.000000000000000000000
##  [2] 12.500000000000000000000
##  [3] 20.833333333333332149095
##  [4] 26.041666666666667850905
##  [5] 26.041666666666667850905
##  [6] 21.701388888888889283635
##  [7] 15.500992063492063266494
##  [8]  9.688120039682539541559
##  [9]  5.382288910934744485814
## [10]  2.691144455467372242907
```

Exercise 7.17

Write a single R command that calculates:

$$\sum_{i=3}^{8} \sum_{j=2}^{9} \frac{i^2}{7+4j}$$

Which simplifies to:

$$\left(\sum_{i=3}^{8} i^2\right)\left(\sum_{j=2}^{9} \frac{1}{7+4j}\right)$$

```
sum(seq(3:8)^2) * sum((seq(2,9)*4+7)^-1)
```

```
## [1] 28.16812873444651543764
```

Exercise 7.18

```
point1<-c(0,0)
point2<-c(3,4)
# part a, L2 norm
sqrt(sum((point2-point1)^2))
```

```
## [1] 5
```

```
# part b, L1 norm
sum(abs(point2-point1))
```

```
## [1] 7
```

Exercise 7.19

Sys.sleep is a suspend function which takes an integer argument, interpreted to be the number of seconds activity should be suspended. For more, see `??Sys.sleep`

Exercise 7.20

```
x<-seq(3,317, by=17)
x
```

```
##  [1]   3  20  37  54  71  88 105 122 139
## [10] 156 173 190 207 224 241 258 275 292
## [19] 309
```

```
length(unique(x))
```

```
## [1] 19
```

Exercise 7.21

```
N<-100
x<-sample(N,N-1,replace=FALSE)
# x<-sample(100,99,replace=FALSE)
1 + order(diff(sort(x)))[length(x)-1]
```

```
## [1] 22
```

```
# Unpacked
# starting with x:
x
```

```
##  [1]  62  26  44 100  48  40  20   9  28
## [10]  49  25  10  66  41  92  43  78  11
## [19]  98  99  50  13  67  69  27   8  94
## [28]  77  68  21  95  74  61  84  70  45
## [37]  72  63  19  87  51  42   4  90  71
## [46]  57  29  59  36  79  53  46  58  91
## [55]  38  55  65  56  14  89   3  52  54
## [64]  85  80   7  33  86   1  83  17  34
## [73]  12  35  15  47  30   2  82  37  81
## [82]  16  76  96  24  60   6  31  75  39
## [91]   5  32  93  18  64  88  73  23  97
```

```
# now, working from the inside out, sort it
sorted_x <- sort(x)
sorted_x
```

```
##  [1]   1   2   3   4   5   6   7   8   9
## [10]  10  11  12  13  14  15  16  17  18
## [19]  19  20  21  23  24  25  26  27  28
## [28]  29  30  31  32  33  34  35  36  37
## [37]  38  39  40  41  42  43  44  45  46
## [46]  47  48  49  50  51  52  53  54  55
## [55]  56  57  58  59  60  61  62  63  64
## [64]  65  66  67  68  69  70  71  72  73
## [73]  74  75  76  77  78  79  80  81  82
## [82]  83  84  85  86  87  88  89  90  91
## [91]  92  93  94  95  96  97  98  99 100
```

```
# the create a vector of differences
differences_x_neighbors <- diff(sorted_x)
differences_x_neighbors
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [19] 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [37] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [55] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [73] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [91] 1 1 1 1 1 1 1 1
```

```
# order() generates a sorted vector of the
#   INDEXES of the DIFF output
#
# ORDER output is :
#   a VECTOR
#   ... of INDEXES
#   ... for the ORDERED values
#   ... of the input vector
#
# in our case, the first (N-1) are "tied"
#   with a value of 1, the Nth however is 2
sorted_indexes_of_differences_x_neighbors <-
    order(differences_x_neighbors)
#
sorted_indexes_of_differences_x_neighbors
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
## [13] 13 14 15 16 17 18 19 20 22 23 24 25
## [25] 26 27 28 29 30 31 32 33 34 35 36 37
## [37] 38 39 40 41 42 43 44 45 46 47 48 49
## [49] 50 51 52 53 54 55 56 57 58 59 60 61
## [61] 62 63 64 65 66 67 68 69 70 71 72 73
## [73] 74 75 76 77 78 79 80 81 82 83 84 85
## [85] 86 87 88 89 90 91 92 93 94 95 96 97
## [97] 98 21
```

```
# pull the last item, which is the 1-based
#   INDEX in X prior to the gap
index_with_largest_difference <-
    sorted_indexes_of_differences_x_neighbors[length(x)-1]
#
index_with_largest_difference
```

```
## [1] 21
```

```
# increment the last item to show the missing one
1 + index_with_largest_difference
```

```
## [1] 22
```

```
# or, to elucidate that we risk
#   conflating INDEXES with VALUES, ...
sorted_x[index_with_largest_difference]
```

```
## [1] 21
```

```
sorted_x[index_with_largest_difference+1]
```

```
## [1] 23
```

```
# missing number is:
sorted_x[index_with_largest_difference]+1
```

```
## [1] 22
```

```
# missing number is:
sorted_x[index_with_largest_difference+1]-1
```

```
## [1] 22
```

Exercises, Chapter 8

- TODO: Finish Exercises

Exercise 8.1

```
reverse = function (x) x[length(x):1]

reverse(1:15)
```

```
##  [1] 15 14 13 12 11 10  9  8  7  6  5  4
## [13]  3  2  1
```

```
x<-sample(100,10,replace=FALSE)
x
```

```
##  [1]  5 96 34 52 25  3 57 45 27 80
```

```
reverse(x)
```

```
##  [1] 80 27 45 57  3 25 52 34 96  5
```

Exercise 8.2

```
my.cos = function(angle=NaN, degrees=FALSE) {
    ifelse(is.na(angle),
            NaN,
            ifelse(degrees,
                    cos(angle),
                    cos(180/pi*angle)
            )
    )
}

my.cos(90,TRUE)
```

```
## [1] -0.4480736161291701269427
```

```
my.cos(90,degrees<-TRUE)
```

```
## [1] -0.4480736161291701269427
```

```
my.cos(pi/2)
```

```
## [1] -0.4480736161291701269427
```

```
my.cos()
```

```
## [1] NaN
```

Exercise 8.3

```
-8^(1/3)
```

```
## [1] -2
```

```
# but
cube.root <- function(x) { (x)^(1/3) }
cube.root(c(-8,8,729,1000000))
```

```
## [1]                      NaN
## [2]  2.0000000000000000000
## [3]  8.9999999999999998223643
## [4] 99.9999999999999971578291
```

```
# and
cube.root <- function(x) { as.numeric(x)^(1/3) }
cube.root(c(-8,8,729,1000000))
```

```
## [1]                      NaN
## [2]  2.0000000000000000000
## [3]  8.9999999999999998223643
## [4] 99.9999999999999971578291
```

```
# so, ...
cube.root <- function(x) { y<-abs(x)^(1/3) ; ifelse(x>=0,y
cube.root(c(-8,8,729,1000000))
```

```
## [1] -2.0000000000000000000
## [2]  2.0000000000000000000
## [3]  8.9999999999999998223643
## [4] 99.9999999999999971578291
```

Exercise 8.4

```
tmean = function (x,k) {
    sorted_x<-sort(x)
    n_from<-k+1
    n_to<-length(x)-k
    mean(sorted_x[n_from:n_to])
}

# case 1
tmean(c(9.4,9.6,9.1,9.5,9.3),1)
```

```
## [1] 9.4000000000000000355271
```

```
# check against :
mean(c(9.4,9.5,9.3))
```

```
## [1] 9.400000000000000355271
```

```
# case 2
tmean(1:18,4)
```

```
## [1] 9.5
```

```
# check against :
mean(5:14)
```

```
## [1] 9.5
```

```
# setup for alternative forms
x<-sample(100,30,replace=FALSE)
x
```

```
##  [1] 19 73 86 68 40 43  7  1 69 87 26 64
## [13]  2 38 13 29 48 83 78 75 17 28 79 30
## [25] 97 39  8 27 22 98
```

```
tmean(x,3)
```

```
## [1] 45.91666666666666429819
```

```
tmean(x,7)
```

```
## [1] 44.9375
```

```
# alternative form #1
# GOTCHA: parenthesis are REQUIRED on the indexes
tmean = function (x,k) {
    y<-sort(x)
    mean(y[(k+1):(length(y)-k)])
}

tmean(c(9.4,9.6,9.1,9.5,9.3),1)
```

```
## [1] 9.400000000000000355271
```

```
tmean(x,3)
```

```
## [1] 45.91666666666666429819
```

```
tmean(x,7)
```

```
## [1] 44.9375
```

```
# alternative form #2
# sort, then subset, then take the mean
tmean = function (x,k) { mean(sort(x)[(k+1):(length(x)-k)]); }
tmean(c(9.4,9.6,9.1,9.5,9.3),1)
```

```
## [1] 9.400000000000000355271
```

```
tmean(x,3)
```

```
## [1] 45.91666666666666429819
```

```
tmean(x,7)
```

```
## [1] 44.9375
```

Exercise 8.5

Exercises, Chapter 9

- TODO: Look for way to convert bases using base R
- TODO: Complete the exercises

Exercise 9.1

```
# setup
targetListOfValues<-c(1:10)
targetListOfValues
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
# before
cos(targetListOfValues)
```

```
##  [1]  0.5403023058681397650105
##  [2] -0.4161468365471424069035
##  [3] -0.9899924966004454152113
##  [4] -0.6536436208636119404858
##  [5]  0.2836621854632262462736
##  [6]  0.9601702866503659672404
##  [7]  0.7539022543433046008587
##  [8] -0.1455000338086135380777
##  [9] -0.9111302618846769396654
## [10] -0.8390715290764524381117
```

```
# set displayed digits to three
options(digits=3)
cos(targetListOfValues)
```

```
##  [1]  0.540 -0.416 -0.990 -0.654  0.284
##  [6]  0.960  0.754 -0.146 -0.911 -0.839
```

```
# set display width to 40 characters
options(width=40)
cos(targetListOfValues)
```

```
##  [1]  0.540 -0.416 -0.990 -0.654  0.284
##  [6]  0.960  0.754 -0.146 -0.911 -0.839
```

```
# force scientific notation
options(scipen=-1000)
cos(targetListOfValues)
```

```
##  [1]  5.40e-01 -4.16e-01 -9.90e-01
##  [4] -6.54e-01  2.84e-01  9.60e-01
##  [7]  7.54e-01 -1.46e-01 -9.11e-01
## [10] -8.39e-01
```

```
# resist scientific notation
options(scipen=1000)
cos(targetListOfValues)
```

```
##  [1]  0.540 -0.416 -0.990 -0.654  0.284
##  [6]  0.960  0.754 -0.146 -0.911 -0.839
```

Exercise 9.2

```
# setup
targetListOfValues<-c(1:10)
options(digits=10)
options(scipen=1000)
options(display=40)
cosValues<-cos(1:10)
cosValues
```

```
##  [1]  0.5403023059 -0.4161468365
##  [3] -0.9899924966 -0.6536436209
##  [5]  0.2836621855  0.9601702867
##  [7]  0.7539022543 -0.1455000338
##  [9] -0.9111302619 -0.8390715291
```

```
# round takes two values
#   the vector of values to be rounded
#   signed integer number of decimal places
round(cosValues,2)
```

```
##  [1]  0.54 -0.42 -0.99 -0.65  0.28  0.96
##  [7]  0.75 -0.15 -0.91 -0.84
```

```
round(cosValues,6)
```

```
##  [1]  0.540302 -0.416147 -0.989992
##  [4] -0.653644  0.283662  0.960170
##  [7]  0.753902 -0.145500 -0.911130
## [10] -0.839072
```

```
round(cosValues,-2)
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0
```

```
round(314.00*cosValues,-2)
```

```
##  [1]  200 -100 -300 -200  100  300  200
##  [8]    0 -300 -300
```

Exercise 9.3

Exercise 9.3.a

Exercise 9.3.b

Exercise 9.3.c

Exercise 9.4

```
# confirming from text
system2("bc",stdout=TRUE,stderr="",input="obase=2;12.75")
```

```
## [1] "1100.1100000"
```

```
# as requested in problem
system2("bc",stdout=TRUE,stderr="",input="obase=2;scale=4;
```

```
## [1] "10.01100000000000"
```

```
# confirming as equivalent value
x<-system2("bc",stdout=TRUE,stderr="",input="obase=2;scale
system2("bc",stdout=TRUE,stderr="",input=paste("obase=10;i
```

```
## [1] "2.37500000000000"
```

```
#`higlighting the problem identified in the text
system2("bc",stdout=TRUE,stderr="",input="obase=2;scale=4;
```

```
## [1] ".01100110011001"
```

```
# confirming as approximate value
x<-system2("bc",stdout=TRUE,stderr="",input="obase=2;scale
system2("bc",stdout=TRUE,stderr="",input=paste("obase=10;i
```

```
## [1] ".39996337890625"
```

Exercise 9.5

```
system2("bc",stdout=TRUE,stderr="",input="obase=8;ibase=2;
```

```
## [1] "574"
```

```
system2("bc",stdout=TRUE,stderr="",input="obase=16;ibase=2
```

```
## [1] "17C"
```

```
system2("bc",stdout=TRUE,stderr="",input="obase=10;ibase=2
```

```
## [1] "380"
```

Exercise 9.6

Exercise 9.7