100 R Vector and Matrix Exercises

This is the R Version of 100 numpy exercises

Install package Magrittr (1 Star)

Load package Magrittr (1 Star)

```
library(magrittr)
```

Print the R version (1 Star)

```
print(version$version.string)
```

```
## [1] "R version 3.3.0 (2016-05-03)"
```

Create a null vector of size 10 (1 Star)

```
rep(NA, 10) %>% print
```

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

How to get the examples of the add (+) function from the command line ? (1 Star)

```
example(`+`)
```

```
##
## +> x <- -1:12
##
## +> x + 1
##  [1]  0  1  2  3  4  5  6  7  8  9 10 11 12 13
##
## +> 2 * x + 3
##  [1]  1  3  5  7  9 11 13 15 17 19 21 23 25 27
##
## +> x %% 2 #-- is periodic
##  [1] 1 0 1 0 1 0 1 0 1 0 1 0 1 0
##
## +> x %/% 5
##  [1] -1  0  0  0  0  0  1  1  1  1  1  2  2  2
```

Create a null vector of size 10 but the fifth value which is 1 (1 Star)

```
z <- rep(NA, 10)
z[5] <- 1
print(z)
```

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

Create a vector with values ranging from 10 to 49 (1 Star)

```
10:49 %>% print
```

```
##  [1] 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
## [24] 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
```

Reverse a vector (first element becomes last) (1 Star)

```
1:10 %>% rev %>% print
```

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

Create a 3x3 matrix with values ranging from 0 to 8 (1 Star)

```
matrix(0:8, ncol = 3, nrow = 3) %>% print
```

```
##      [,1] [,2] [,3]
## [1,]    0    3    6
## [2,]    1    4    7
## [3,]    2    5    8
```

Find indices of non-zero elements from (1,2,0,0,4,0) (1 Star)

```
c(1,2,0,0,4,0) %>% as.logical %>% which %>% print
```

```
## [1] 1 2 5
```

Create a 3x3 identity matrix (1 Star)

```
diag(3) %>% print
```

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

Create a 3x3x3 array with random values (1 Star)

```
runif(3^3) %>% array(c(3, 3, 3)) %>% print
```

```
## , , 1
##
##           [,1]       [,2]       [,3]
## [1,] 0.7142955 0.36102309 0.62756149
## [2,] 0.1667387 0.13370829 0.91158593
## [3,] 0.4585626 0.06159756 0.02060076
##
## , , 2
##
##           [,1]       [,2]      [,3]
```

```
## [1,] 0.5447111 0.08364432 0.8719061
## [2,] 0.7398042 0.78339599 0.9002058
## [3,] 0.2400276 0.86527716 0.9724733
##
## , , 3
##
##           [,1]       [,2]      [,3]
## [1,] 0.1268534 0.16016971 0.9553357
## [2,] 0.1921771 0.08059356 0.8391543
## [3,] 0.9269792 0.27991209 0.9081627
```

Create a 10x10 array with random values and find the minimum and maximum values (1 Star)

```
z <- array(runif(10^2), c(10,10))
min(z) %>% print
```

```
## [1] 0.003423488
```

```
max(z) %>% print
```

```
## [1] 0.9942363
```

Create a random vector of size 30 and find the mean value (1 Star)

```
runif(30) %>% mean %>% print
```

```
## [1] 0.5288562
```

Create a 2d array with 1 on the border and 0 inside (1 Star)

```
z <- array(1, c(4, 5))
z[2:(nrow(z) - 1), 2:(ncol(z) - 1)] <- 0
print(z)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    1    1    1    1
## [2,]    1    0    0    0    1
## [3,]    1    0    0    0    1
## [4,]    1    1    1    1    1
```

What is the result of the following expression ? (1 Star)

```
0 * NaN
```

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

```
NA == NA
```

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

```
NA == NaN
```

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

```
Inf > NA
```

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

```
NaN - NaN
```

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

```
0.3 == 3 * 0.1
```

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

Create a 5x5 matrix with values 1,2,3,4 just below the diagonal (1 Star)

```
z <- diag(0, ncol = 5, nrow = 5)
z[row(z) - 1 == col(z)] <- 1:4
print(z)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0    0    0    0
## [2,]    1    0    0    0    0
## [3,]    0    2    0    0    0
## [4,]    0    0    3    0    0
## [5,]    0    0    0    4    0
```

Create a 8x8 matrix and fill it with a checkerboard pattern (1 Star)

```
z <- array(0, c(8, 8))
z[1:nrow(z) %% 2 != 0, 1:ncol(z) %% 2 == 0] <- 1
z[1:nrow(z) %% 2 == 0, 1:ncol(z) %% 2 != 0] <- 1
print(z)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    0    1    0    1    0    1    0    1
## [2,]    1    0    1    0    1    0    1    0
## [3,]    0    1    0    1    0    1    0    1
## [4,]    1    0    1    0    1    0    1    0
## [5,]    0    1    0    1    0    1    0    1
## [6,]    1    0    1    0    1    0    1    0
## [7,]    0    1    0    1    0    1    0    1
## [8,]    1    0    1    0    1    0    1    0
```

Consider a (6,7,8) shape array, what is the index (x,y,z) of the 100th element ?

```
z <- array(runif(6 * 7 * 8), c(6, 7, 8))
which(z == z[100], arr.ind = T)
```

```
##      dim1 dim2 dim3
## [1,]    4    3    3
```

Create a checkerboard 8x8 matrix using the rep function (1 Star)

```
odd <- rep(0:1, 4)
evn <- rep(1:0, 4)
z <- array(c(odd, evn), c(8, 8))
print(z)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    0    1    0    1    0    1    0    1
## [2,]    1    0    1    0    1    0    1    0
## [3,]    0    1    0    1    0    1    0    1
## [4,]    1    0    1    0    1    0    1    0
## [5,]    0    1    0    1    0    1    0    1
## [6,]    1    0    1    0    1    0    1    0
## [7,]    0    1    0    1    0    1    0    1
## [8,]    1    0    1    0    1    0    1    0
```

Normalize a 5x5 random matrix (1 Star)

```
runif(5 * 5) %>% matrix(nrow = 5, ncol = 5) %>% scale %>% print
```

```
##            [,1]       [,2]       [,3]        [,4]       [,5]
## [1,]  1.5213613 -0.7626726 -0.6746466  1.34215650 -1.6770739
## [2,] -0.3933881  0.2214633  1.4824910 -1.45340208  0.1426949
## [3,] -0.8598246  1.4550611  0.5270251 -0.06132372  0.9854621
## [4,]  0.4805213  0.1890303 -0.9660807  0.27011562  0.1240077
## [5,] -0.7486699 -1.1028822 -0.3687887 -0.09754632  0.4249092
## attr(,"scaled:center")
## [1] 0.4355181 0.5307829 0.3510867 0.3875995 0.6696882
## attr(,"scaled:scale")
## [1] 0.3587730 0.3019312 0.1715997 0.2459070 0.1042865
```

Multiply a 5x3 matrix by a 3x2 matrix (real matrix product) (1 Star)

```
matrix(1, 5, 3) %*% matrix(1, 3, 2) %>% print
```

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

Given a 1D array, negate all elements which are between 3 and 8, in place. (1 Star)

```r
z <- 1:15
ifelse(z > 3 & z <= 8, -z, z) %>% print
```

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

Create a 5x5 matrix with row values ranging from 0 to 4 (1 Star)

```r
matrix(0:4, 5, 5)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0    0    0    0
## [2,]    1    1    1    1    1
## [3,]    2    2    2    2    2
## [4,]    3    3    3    3    3
## [5,]    4    4    4    4    4
```

Generate 5 vectors and use them to build a matrix (2 Stars)

```r
vecs <- paste0("vec", 1:5)

for (i in 1:5){
  assign(vecs[i], runif(5))
}

sapply(vecs, get)
```

```
##            vec1      vec2      vec3       vec4      vec5
## [1,] 0.09372989 0.4701712 0.4729154 0.26849760 0.9857478
## [2,] 0.87769213 0.3216424 0.3442144 0.34601346 0.6102098
## [3,] 0.93663393 0.9798213 0.6752808 0.05875189 0.4580793
## [4,] 0.53918680 0.6897475 0.5357932 0.17639150 0.3051970
## [5,] 0.08407286 0.2000597 0.3393981 0.38779564 0.1745104
```

Create a vector of size 10 with values ranging from 0 to 1, both excluded (2 Stars)

```r
z <- seq(0, 1, length.out = 12)
z[c(-1, -length(z))] %>% print
```

```
##  [1] 0.09090909 0.18181818 0.27272727 0.36363636 0.45454545 0.54545455
##  [7] 0.63636364 0.72727273 0.81818182 0.90909091
```

Create a random vector of size 10 and sort it (2 Stars)

```r
runif(10) %>% sort %>% print
```

```
##  [1] 0.02550455 0.15237801 0.38855476 0.41283482 0.45429005 0.60311907
##  [7] 0.73874044 0.74611576 0.84713689 0.87321635
```

How to sum a small array faster without sum ? (2 Stars)

```r
1:10 %>% {Reduce(`+`, .)} %>% print
```

```
## [1] 55
```

Consider two random array A anb B, check if they are equal (2 Stars)

```r
a <- replicate(5, runif(5)) %>% as.array
b <- replicate(5, runif(5)) %>% as.array
all(a == b) %>% print
```

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

Make an array immutable (read-only) (2 Stars)

```r
z <- array(runif(3), c(3, 4))
lockBinding("z", env = .GlobalEnv)
try(z <- array(runif(3), c(3, 4))) %>% print
```

```
## [1] "Error in try(z <- array(runif(3), c(3, 4))) : \n  cannot change value of locked binding for 'z'
## attr(,"class")
## [1] "try-error"
## attr(,"condition")
## <simpleError in doTryCatch(return(expr), name, parentenv, handler): cannot change value of locked bi
```

```r
unlockBinding("z", env = .GlobalEnv)
```

Consider a random 10x2 matrix representing cartesian coordinates, convert them to polar coordinates (2 Stars)

```r
z <- replicate(2, runif(10))
x <- z[,1]
y <- z[,2]
r = sqrt(x^2 + y^2) %>% print
```

```
##  [1] 0.8228424 0.7349186 0.5898940 0.5231623 1.0972820 0.9973436 0.8349745
##  [8] 1.0652835 0.4548002 0.7806865
```

```r
t = atan2(y, x) %>% print
```

```
##  [1] 0.9275352 0.3816021 1.2025011 0.6748579 0.4418061 0.4298307 0.5536284
##  [8] 0.7484951 0.7640060 0.2383538
```

Create random vector of size 10 and replace the maximum value by 0 (2 Stars)

```r
z <- runif(10)
z[max(z) == z] <- 0
print(z)
```

```
##  [1] 0.9466015 0.3725144 0.3422989 0.1597569 0.5820692 0.1996449 0.0000000
##  [8] 0.7996576 0.6179456 0.4292679
```

Create a structured array with x and y coordinates covering the [0,1]x[0,1] area (2 Stars)

```
z <- array(NA, dim = c(10,10))

x <- seq(0, 1, length.out = 10)
outer(x, x, FUN = paste, sep = ", ")
```

```
##       [,1]                  [,2]
##  [1,] "0, 0"                "0, 0.111111111111111"
##  [2,] "0.111111111111111, 0" "0.111111111111111, 0.111111111111111"
##  [3,] "0.222222222222222, 0" "0.222222222222222, 0.111111111111111"
##  [4,] "0.333333333333333, 0" "0.333333333333333, 0.111111111111111"
##  [5,] "0.444444444444444, 0" "0.444444444444444, 0.111111111111111"
##  [6,] "0.555555555555556, 0" "0.555555555555556, 0.111111111111111"
##  [7,] "0.666666666666667, 0" "0.666666666666667, 0.111111111111111"
##  [8,] "0.777777777777778, 0" "0.777777777777778, 0.111111111111111"
##  [9,] "0.888888888888889, 0" "0.888888888888889, 0.111111111111111"
## [10,] "1, 0"                "1, 0.111111111111111"
##       [,3]
##  [1,] "0, 0.222222222222222"
##  [2,] "0.111111111111111, 0.222222222222222"
##  [3,] "0.222222222222222, 0.222222222222222"
##  [4,] "0.333333333333333, 0.222222222222222"
##  [5,] "0.444444444444444, 0.222222222222222"
##  [6,] "0.555555555555556, 0.222222222222222"
##  [7,] "0.666666666666667, 0.222222222222222"
##  [8,] "0.777777777777778, 0.222222222222222"
##  [9,] "0.888888888888889, 0.222222222222222"
## [10,] "1, 0.222222222222222"
##       [,4]
##  [1,] "0, 0.333333333333333"
##  [2,] "0.111111111111111, 0.333333333333333"
##  [3,] "0.222222222222222, 0.333333333333333"
##  [4,] "0.333333333333333, 0.333333333333333"
##  [5,] "0.444444444444444, 0.333333333333333"
##  [6,] "0.555555555555556, 0.333333333333333"
##  [7,] "0.666666666666667, 0.333333333333333"
##  [8,] "0.777777777777778, 0.333333333333333"
##  [9,] "0.888888888888889, 0.333333333333333"
## [10,] "1, 0.333333333333333"
##       [,5]
##  [1,] "0, 0.444444444444444"
##  [2,] "0.111111111111111, 0.444444444444444"
##  [3,] "0.222222222222222, 0.444444444444444"
##  [4,] "0.333333333333333, 0.444444444444444"
##  [5,] "0.444444444444444, 0.444444444444444"
##  [6,] "0.555555555555556, 0.444444444444444"
##  [7,] "0.666666666666667, 0.444444444444444"
##  [8,] "0.777777777777778, 0.444444444444444"
##  [9,] "0.888888888888889, 0.444444444444444"
```

```
## [10,] "1, 0.444444444444444"
##       [,6]
##  [1,] "0, 0.555555555555556"
##  [2,] "0.111111111111111, 0.555555555555556"
##  [3,] "0.222222222222222, 0.555555555555556"
##  [4,] "0.333333333333333, 0.555555555555556"
##  [5,] "0.444444444444444, 0.555555555555556"
##  [6,] "0.555555555555556, 0.555555555555556"
##  [7,] "0.666666666666667, 0.555555555555556"
##  [8,] "0.777777777777778, 0.555555555555556"
##  [9,] "0.888888888888889, 0.555555555555556"
## [10,] "1, 0.555555555555556"
##       [,7]
##  [1,] "0, 0.666666666666667"
##  [2,] "0.111111111111111, 0.666666666666667"
##  [3,] "0.222222222222222, 0.666666666666667"
##  [4,] "0.333333333333333, 0.666666666666667"
##  [5,] "0.444444444444444, 0.666666666666667"
##  [6,] "0.555555555555556, 0.666666666666667"
##  [7,] "0.666666666666667, 0.666666666666667"
##  [8,] "0.777777777777778, 0.666666666666667"
##  [9,] "0.888888888888889, 0.666666666666667"
## [10,] "1, 0.666666666666667"
##       [,8]
##  [1,] "0, 0.777777777777778"
##  [2,] "0.111111111111111, 0.777777777777778"
##  [3,] "0.222222222222222, 0.777777777777778"
##  [4,] "0.333333333333333, 0.777777777777778"
##  [5,] "0.444444444444444, 0.777777777777778"
##  [6,] "0.555555555555556, 0.777777777777778"
##  [7,] "0.666666666666667, 0.777777777777778"
##  [8,] "0.777777777777778, 0.777777777777778"
##  [9,] "0.888888888888889, 0.777777777777778"
## [10,] "1, 0.777777777777778"
##       [,9]                                  [,10]
##  [1,] "0, 0.888888888888889"                "0, 1"
##  [2,] "0.111111111111111, 0.888888888888889" "0.111111111111111, 1"
##  [3,] "0.222222222222222, 0.888888888888889" "0.222222222222222, 1"
##  [4,] "0.333333333333333, 0.888888888888889" "0.333333333333333, 1"
##  [5,] "0.444444444444444, 0.888888888888889" "0.444444444444444, 1"
##  [6,] "0.555555555555556, 0.888888888888889" "0.555555555555556, 1"
##  [7,] "0.666666666666667, 0.888888888888889" "0.666666666666667, 1"
##  [8,] "0.777777777777778, 0.888888888888889" "0.777777777777778, 1"
##  [9,] "0.888888888888889, 0.888888888888889" "0.888888888888889, 1"
## [10,] "1, 0.888888888888889"                "1, 1"
```

Given two arrays, X and Y, construct the Cauchy matrix C (Cij = 1/(xi - yj))

```
x <- 1:8
y <- x + 0.5
cmt <- 1.0 / outer(x, y, FUN = "-")
print(det(cmt))
```

```
## [1] 3638.164
```