VECTORS and MATRICES

VECTORS

Vectors are the most basic R data objects and there are six types of atomic vectors. They are logical, integer, double, complex, character and raw.

```
# Atomic vector of type character.
print("abc");
 # Atomic vector of type double.
 print(12.5)
# Atomic vector of type integer.
print(63L)
# Atomic vector of type logical.
 print(TRUE)
# Atomic vector of type complex.
print(2+3i)
 # Atomic vector of type raw.
print(charToRaw('hello'))
```

```
[1] "abc"
[1] 12.5
[1] 63
[1] TRUE
[1] 2+3i
[1] 68 65 6c 6c 6f
```

VECTORS

Multiple Elements Vector

Using colon operator with numeric data

```
#Creating a sequence from 5 to 13.
 v <- 5:13
 print(v)
# Creating a sequence from 6.6 to 12.6.
v <- 6.6:12.6
print(v)
# If the final element specified does not
belong to the sequence then it is discarded.
v <- 3.8:11.4
print(v)
```

```
[1] 5 6 7 8 9 10 11 12 13
[1] 6.6 7.6 8.6 9.6 10.6 11.6 12.6
[1] 3.8 4.8 5.8 6.8 7.8 8.8 9.8 10.8
```

VECTORS

```
# Create vector with elements from 5 to 9 incrementing by 0.4.
print(seq(5, 9, by = 0.4))
```

```
[1] 5.0 5.4 5.8 6.2 6.6 7.0 7.4 7.8 8.2 8.6 9.0
```

VECTORS

Using the c() function

The non-character values are coerced to character type if one of the elements is a character.

```
# The logical and numeric values are converted to characters.
s <- c('apple','red',5,TRUE)
 print(s)
```

```
[1] "apple" "red" "5" "TRUE"
```

VECTORS

Accessing Vector Elements

Elements of a Vector are accessed using indexing. The **[ ] brackets** are used for indexing. Indexing starts with position 1. Giving a negative value in the index drops that element from result. **TRUE**, **FALSE** or **0** and **1** can also be used for indexing.

```
# Accessing vector elements using position.
t <- c("Sun","Mon","Tue","Wed","Thurs","Fri","Sat")
u <- t[c(2,3,6)]
print(u)
# Accessing vector elements using logical indexing.
 v <- t[c(TRUE,FALSE,FALSE,FALSE,FALSE,TRUE,FALSE)]
 print(v) # Accessing vector elements using negative indexing.
x <- t[c(-2,-5)]
 print(x)
# Accessing vector elements using 0/1 indexing.
 y <- t[c(0,0,0,0,0,0,1)]
 print(y)
```

```
[1] "Mon" "Tue" "Fri"
[1] "Sun" "Fri"
[1] "Sun" "Tue" "Wed" "Fri" "Sat"
[1] "Sun"
```

VECTORS

Vector Manipulation: Vector arithmetic

Two vectors of same length can be added, subtracted, multiplied or divided giving the result as a vector output.

```
# Create two vectors.
 v1 <- c(3,8,4,5,0,11)
 v2 <- c(4,11,0,8,1,2)
# Vector addition.
add.result <- v1+v2 print(add.result)
 # Vector subtraction.
sub.result <- v1-v2
print(sub.result)
 # Vector multiplication.
multi.result <- v1*v2
print(multi.result)
# Vector division.
 divi.result <- v1/v2
 print(divi.result)
```

```
[1] 7 19 4 13 1 13
[1] -1 -3 4 -3 -1 9
[1] 12 88 0 40 0 22
[1] 0.7500000 0.7272727 Inf 0.6250000 0.0000000 5.5000000
```

VECTORS

Vector Element Recycling

If we apply arithmetic operations to two vectors of unequal length, then the elements of the shorter vector are recycled to complete the operations.

```
v1 <- c(3,8,4,5,0,11)
v2 <- c(4,11)
# V2 becomes c(4,11,4,11,4,11)
 add.result <- v1+v2
print(add.result)
 sub.result <- v1-v2
print(sub.result)
```

```
[1] 7 19 8 16 4 22
[1] -1 -3 0 -6 -4 0
```

VECTORS

Vector Element Sorting

Elements in a vector can be sorted using the **sort()** function.

```
v <- c(3,8,4,5,0,11, -9, 304)
# Sort the elements of the vector.
sort.result <- sort(v)
print(sort.result)
 # Sort the elements in the reverse order.
 revsort.result <- sort(v, decreasing = TRUE)
print(revsort.result)
# Sorting character vectors.
 v <- c("Red","Blue","yellow","violet")
sort.result <- sort(v)
print(sort.result)
# Sorting character vectors in reverse order.
revsort.result <- sort(v, decreasing = TRUE)
 print(revsort.result)
```

```
[1] -9 0 3 4 5 8 11 304
[1] 304 11 8 5 4 3 0 -9
[1] "Blue" "Red" "violet" "yellow"
[1] "yellow" "violet" "Red" "Blue"
```

MATRICES

Matrices are the R objects in which the elements are arranged in a two-dimensional rectangular layout. They contain elements of the same atomic types.

MATRICES

A Matrix is created using the **matrix()** function.

Syntax

The basic syntax for creating a matrix in R is −

matrix(data, nrow, ncol, byrow, dimnames)

Following is the description of the parameters used −

- **data** is the input vector which becomes the data elements of the matrix.
- **nrow** is the number of rows to be created.
- **ncol** is the number of columns to be created.
- **byrow** is a logical clue. If TRUE then the input vector elements are arranged by row.
- **dimname** is the names assigned to the rows and columns.

MATRICES

```
# Elements are arranged sequentially by row.
M <- matrix(c(3:14), nrow = 4, byrow = TRUE)
print(M)
```

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

MATRICES

```
# Elements are arranged sequentially by column.
N <- matrix(c(3:14), nrow = 4, byrow = FALSE)
print(N)
```

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

MATRICES

```
A = matrix(
  # Taking sequence of elements
  c(1, 2, 3, 4, 5, 6, 7, 8, 9),
   # No of rows
  nrow = 3,
    # No of columns
  ncol = 3,
    # By default matrices are in column-wise order
  # So this parameter decides how to arrange the matrix
  byrow = TRUE     )
  # Naming rows
rownames(A) = c("a", "b", "c")
  # Naming columns
colnames(A) = c("c", "d", "e")
  cat("The 3x3 matrix:\n")
print(A)
```

```
The 3x3 matrix:
  c d e
a 1 2 3
b 4 5 6
c 7 8 9
```

MATRICES

```
# Define the column and row names.
rownames = c("row1", "row2", "row3", "row4")
colnames = c("col1", "col2", "col3")
P <- matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames =
list(rownames, colnames))
print(P)
```

```
     col1 col2 col3
row1    3    4    5
row2    6    7    8
row3    9   10   11
row4   12   13   14
```

MATRICES

Accessing Elements of a Matrix

Elements of a matrix can be accessed by using the column and row index of the element. We consider the matrix P above to find the specific elements below.

MATRICES

```
# Define the column and row names.
 rownames = c("row1", "row2", "row3", "row4")
colnames = c("col1", "col2", "col3")
 # Create the matrix.
P <- matrix(c(3:14), nrow = 4, byrow = TRUE,
dimnames = list(rownames, colnames))
 # Access the element at 3rd column and 1st row.
print(P[1,3])
# Access the element at 2nd column and 4th row. print(P[4,2])
 # Access only the 2nd row. print(P[2,])
 # Access only the 3rd column. print(P[,3])
```

```
[1] 5
[1] 13
col1 col2 col3
   6    7    8
row1 row2 row3 row4
   5    8   11   14
```

MATRICES

```
cat("Accessing the first three rows and the first two columns\n")
print(A[1:3, 1:2])
```

MATRIX COMPUTATIONS

Various mathematical operations are performed on the matrices using the R operators. The result of the operation is also a matrix.
The dimensions (number of rows and columns) should be same for the matrices involved in the operation.

MATRIX COMPUTATIONS

```
#Create two 2x3 matrices.
matrix1 <- matrix(c(3, 9, -1, 4, 2, 6), nrow = 2)
print(matrix1)
 matrix2 <- matrix(c(5, 2, 0, 9, 3, 4), nrow = 2)
print(matrix2)
# Add the matrices.
result <- matrix1 + matrix2
cat("Result of addition","\n")
print(result)
# Subtract the matrices
result <- matrix1 - matrix2
cat("Result of subtraction","\n")
print(result)
```

MATRIX COMPUTATIONS

```
     [,1] [,2] [,3]
[1,]    3   -1    2
[2,]    9    4    6
     [,1] [,2] [,3]
[1,]    5    0    3
[2,]    2    9    4
Result of addition
     [,1] [,2] [,3]
[1,]    8   -1    5
[2,]   11   13   10
Result of subtraction
     [,1] [,2] [,3]
[1,]   -2   -1   -1
```

MATRIX COMPUTATIONS

```
# Create two 2x3 matrices.
 matrix1 <- matrix(c(3, 9, -1, 4, 2, 6), nrow = 2)
print(matrix1)
 matrix2 <- matrix(c(5, 2, 0, 9, 3, 4), nrow = 2)
 print(matrix2)
# Multiply the matrices.
result <- matrix1 * matrix2
cat("Result of multiplication","\n")
print(result)
 # Divide the matrices
 result <- matrix1 / matrix2
cat("Result of division","\n")
print(result)
```

MATRIX COMPUTATIONS

```
[,1] [,2] [,3]
[1,] 3 -1 2
[2,] 9 4 6
[,1] [,2] [,3]
[1,] 5 0 3
[2,] 2 9 4
Result of multiplication
[,1] [,2] [,3]
[1,] 15 0 6
[2,] 18 36 24
Result of division
[,1] [,2] [,3]
[1,] 0.6 -Inf 0.6666667
[2,] 4.5 0.4444444 1.5000000
```

MATRIX COMPUTATIONS

```
# Matrix having 3 rows and 3 columns
# filled by a single constant 5
print(matrix(5, 3, 3))
```

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

MATRIX COMPUTATIONS

```
# Diagonal matrix having 3 rows and 3 columns
# filled by array of elements (5, 3, 3)
print(diag(c(5, 3, 3), 3, 3))
```

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

MATRIX COMPUTATIONS

```
# Identity matrix having
# 3 rows and 3 columns
print(diag(1, 3, 3))
```

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

MATRIX COMPUTATIONS

```
# Create a 3x3 matrix
A = matrix( c(1, 2, 3, 4, 5, 6, 7, 8, 9),
  nrow = 3,
  ncol = 3,
  byrow = TRUE      )
cat("The 3x3 matrix:\n")
print(A)
 cat("Dimension of the matrix:\n")
print(dim(A))
 cat("Number of rows:\n")
print(nrow(A))
 cat("Number of columns:\n")
print(ncol(A))
 cat("Number of elements:\n")
print(length(A))
# OR print(prod(dim(A)))
```

```
> cat("Dimension of the matrix:\n")
Dimension of the matrix:
> print(dim(A))
[1] 3 3
> cat("Number of rows:\n")
Number of rows:
> print(nrow(A))
[1] 3
> cat("Number of columns:\n")
Number of columns:
> print(ncol(A))
[1] 3
> cat("Number of elements:\n")
Number of elements:
> print(length(A))
[1] 9
> # OR print(prod(dim(A)))
> print(prod(dim(A)))
[1] 9
```

Matrix Concatenation

Matrix concatenation refers to the merging of rows or columns of an existing matrix.

Concatenation of a row:

The concatenation of a row to a matrix is done using **rbind()**.

The concatenation of a column to a matrix is done using **cbind()**.

Matrix Concatenation

```
# Create a 3x3 matrix
A = matrix( c(1, 2, 3, 4, 5, 6, 7, 8, 9),
  nrow = 3,
  ncol = 3,
  byrow = TRUE    )
cat("The 3x3 matrix:\n")
print(A)
 # Creating another 1x3 matrix
B = matrix(  c(10, 11, 12),   nrow = 1,   ncol = 3 )
cat("The 1x3 matrix:\n")
print(B)
 # Add a new row using rbind()
C = rbind(A, B)
 cat("After concatenation of a row:\n")
print(C)
```

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

The 1x3 matrix:
     [, 1] [, 2] [, 3]
[1, ]   10   11   12

After concatenation of a row:
     [, 1] [, 2] [, 3]
[1, ]    1    2    3
[2, ]    4    5    6
[3, ]    7    8    9
[4, ]   10   11   12
```

Matrix Concatenation

```
 Creating another 3x1 matrix
B = matrix(
  c(10, 11, 12),
  nrow = 3,   ncol = 1,   byrow = TRUE
)
cat("The 3x1 matrix:\n")
print(B)
 # Add a new column using cbind()
C = cbind(A, B)

cat("After concatenation of a column:\n")
print(C)
```

```
The 3x3 matrix:
     [, 1] [, 2] [, 3]
[1, ]    1    2    3
[2, ]    4    5    6
[3, ]    7    8    9

The 3x1 matrix:
     [, 1]
[1, ]   10
[2, ]   11
[3, ]   12

After concatenation of a column:
     [, 1] [, 2] [, 3] [, 4]
[1, ]    1    2    3   10
[2, ]    4    5    6   11
[3, ]    7    8    9   12
```

MATRIX COMPUTATIO

```
# Dimension inconsistency in metrics concatenation
 # Create a 3x3 matrix
A = matrix(
  c(1, 2, 3, 4, 5, 6, 7, 8, 9),
  nrow = 3,
ncol = 3,
  byrow = TRUE )
cat("The 3x3 matrix:\n")
print(A)
 # Creating another 1x3 matrix
B = matrix(c(10, 11, 12),
  nrow = 1,  ncol = 3, )
cat("The 1x3 matrix:\n")
print(B)
```

```
The 3x3 matrix:
     [, 1] [, 2] [, 3]
[1, ]    1    2    3
[2, ]    4    5    6
[3, ]    7    8    9

The 1x3 matrix:
     [, 1] [, 2] [, 3]
[1, ]   10   11   12

Error in cbind(A, B) : number of rows of matrices must match (s
```

```
# This will give an error
# because of dimension inconsistency
C = cbind(A, B)

cat("After concatenation of a column:\n")
print(C)
```

Row Deletion

```
#row deletion in metrics
 # Create a 3x3 matrix
A = matrix(
  c(1, 2, 3, 4, 5, 6, 7, 8, 9),
  nrow = 3,
  ncol = 3,
  byrow = TRUE
)
cat("Before deleting the 2nd row\n")
print(A)
 # 2nd-row deletion
A = A[-2, ]
 cat("After deleted the 2nd row\n")
print(A)
```

```
Before deleting the 2nd row
     [, 1] [, 2] [, 3]
[1, ]    1    2    3
[2, ]    4    5    6
[3, ]    7    8    9

After deleted the 2nd row
     [, 1] [, 2] [, 3]
[1, ]    1    2    3
[2, ]    7    8    9
```

```
# Editing the 3rd rows and 3rd column element
# from 9 to 30
# by direct assignments
A[3, 3] = 30

cat("After edited the matrix\n")
print(A)
```

Column Deletion

```r
# column deletion in metrics
 # Create a 3x3 matrix
A = matrix(
  c(1, 2, 3, 4, 5, 6, 7, 8, 9),
  nrow = 3,
  ncol = 3,
  byrow = TRUE     )
cat("Before deleting the 2nd
column\n")
print(A)
 # 2nd-row deletion
A = A[, -2]

cat("After deleted the 2nd column\n")
print(A)
```

```
Before deleting the 2nd column
     [, 1] [, 2] [, 3]
[1, ]    1    2    3
[2, ]    4    5    6
[3, ]    7    8    9

After deleted the 2nd column
     [, 1] [, 2]
[1, ]    1    3
[2, ]    4    6
[3, ]    7    9
```

cat is valid only for atomic types (logical, integer, real, complex, character)
and names. It means you cannot call cat on a non-empty list or any type of object.
In practice it simply converts arguments to characters and concatenates
so you can think of something like as.character() %>% paste().

cat returns an object of class NULL.

print returns a character vector:


Thank
You