R DataStructures

Data Types

Scalar types

numeric,
character,
logical,
complex

Complex types

vectors,
matrices, arrays
data frames,
lists,
factors

Scalar Types

a = 10

b = “abc”

c = TRUE

d = 10 + i 20

Arithmetic Operators

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
^ or **	exponentiation
x %% y	modulus (x mod y) 5%%2 is 1
x %/% y	integer division 5%/%2 is 2

Logical Operators

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x
x | y	x OR y
x & y	x AND y
isTRUE(x)	test if x is TRUE

Numeric Functions

Function	Description
abs(*x***)**	absolute value
sqrt(*x***)**	square root
ceiling(*x***)**	ceiling(3.475) is 4
floor(*x***)**	floor(3.475) is 3
trunc(*x***)**	trunc(5.99) is 5
round(*x***, digits=***n***)**	round(3.475, digits=2) is 3.48
signif(*x***, digits=***n***)**	signif(3.475, digits=2) is 3.5
cos(*x***), sin(***x***), tan(***x***)**	also acos(*x*), cosh(*x*), acosh(*x*), etc.
log(*x***)**	natural logarithm
log10(*x***)**	common logarithm
exp(*x***)**	e^*x*

Character Functions

Function	Description
substr(*x***, start=***n1***, stop=***n2***)**	Extract or replace substrings in a character vector. x <- "abcdef" substr(x, 2, 4) is "bcd" substr(x, 2, 4) <- "22222" is "a222ef"
grep(*pattern***,** *x* **, ignore.case=**FALSE**, fixed=**FALSE**)**	Search for *pattern* in *x*. If fixed =FALSE then *pattern* is a regular expression. If fixed=TRUE then *pattern* is a text string. Returns matching indices. grep("A", c("b","A","c"), fixed=TRUE) returns 2

sub(*pattern***,** *replacement***,** *x***,** **ignore.case** =FALSE**,** **fixed=**FALSE**)**	Find *pattern* in *x* and replace with *replacement* text. If fixed=FALSE then *pattern* is a regular expression. If fixed = T then *pattern* is a text string. sub("\\s",".","Hello There") returns "Hello.There"
strsplit(*x***,** *split***)**	Split the elements of character vector *x* at *split*. strsplit("abc", "") returns 3 element vector "a","b","c"
paste(..., sep="")	Concatenate strings after using *sep* string to seperate them. paste("x",1:3,sep="") returns c("x1","x2" "x3") paste("x",1:3,sep="M") returns c("xM1","xM2" "xM3") paste("Today is", date())

toupper(*x***)**	Uppercase
tolower(*x***)**	Lowercase

Type Check & Conversion

Type Check

- Use is.*foo* to test for data type *foo*. Returns TRUE or FALSE
- is.numeric(), is.character(), is.vector(), is.matrix(), is.data.frame()

Type Conversion

- Type conversions in R work as you would expect. For example, adding a character string to a numeric vector converts all the elements in the vector to character.

- Use as.*foo* to explicitly convert it.

- as.numeric(), as.character(), as.vector(), as.matrix(), as.data.frame)

Vectors

Vectors

a. Creating vectors

-concatenate function: *c()*, combines specified values in a vector

```
> v=c(1,2,3,20)
```

-colon operator: :, generates an ordered sequence incremented by 1

```
> v=1:4
```

-sequence function: *seq()*, generates an ordered sequence incremented by the specified value or a sequence of the specified length

```
> v=seq(from=5,to=6,by=0.1)
> v=seq(from=-10,to=-5,length.out=10)
```

-repeat function: *rep()*, generates a vector of specified length containing the same value in each element of the vector

```
> v=rep(x=4,by=5)
```

-numeric function: *numeric()*, generates a vector of specified length filled with 0's

```
> v=numeric(length=5)
```

-vector function: *vector()*, generates a vector of specified length filled with FALSE's

```
> v=vector(length=5)
```

Vectors

b. Useful vector functions

-mathematical operators: +, - , *, /

-logical operators: <, >, <=, >=, ==, !=

-*length()*: returns the length of the vector

-*max()*: returns the maximum value contained in the vector

-*min()*: returns the minimum value contained in the vector

-*sum()*: returns the sum of the values in the vector

-*cumsum()*: returns the cumulative sum for each element of the vector

-*mean()*: returns the mean of the vector

-*range()*: returns the minimum and maximum values

-*var()*: returns the variance of the vector

-*sd()*: returns the standard deviation of the vector

-*sort()*: returns a sorted version of the vector

-*order()*: returns the numerical indices of vector elements in sorted order

Vectors

c. Vector indexing and subsetting

Because vectors are an ordered list, a single element or subset of elements can be referred to using square brackets, *[ ]*, and a numerical index.

```
> v=c(1,3,5,9,13)
> v[1]
[1] 1
> v[4]
[1] 9
> v[c(1,3,5)]
[1] 1 5 13
> v[-3]
[1] 1 3 9 13
```

Vectors

An alternative means of indexing is a vector of logical values.

```
> v=c(1,3,5,9,13)
> v>3
[1] FALSE FALSE TRUE TRUE TRUE
>v[v>3]
[1] 5 9 13
```

The which() function creates numerical indices from a logical vector.

```
> v=c(1,3,5,9,13)
> v>3
[1] FALSE FALSE TRUE TRUE TRUE
> which(v>3)
[1] 3 4 5
```

Vectors

Note that you cannot delete an element from a vector, but you can reassign a subset of a vector to the same variable.

```
> v=c(1,3,5,9,13)
> v
[1] 1 3 5 9 13
> v=v[v>3]
> v
[1] 5 9 13
```

Matrices

Matrices

All columns in a matrix must have the same type and the same length.

The general format is:

mymatrix <- **matrix(***vector*, **nrow=***r*, **ncol=***c*, **byrow=***FALSE*,**dimnames=list(***char_vector_rownames*, *char_vector_colnames***))**

byrow=TRUE indicates that the matrix should be filled by rows.

byrow=FALSE indicates that the matrix should be filled by columns (the default).

dimnames provides optional labels for the columns and rows.

Matrices

```
# generates 5 x 4 numeric matrix
   y<-matrix(1:20, nrow=5,ncol=4)
# another example
   cells <- c(1,26,24,68)
   rnames <- c("R1", "R2")
   cnames <- c("C1", "C2")
   mymatrix <- matrix(cells, nrow=2, ncol=2,
   byrow=TRUE, dimnames=list(rnames, cnames))
#Identify rows, columns or elements using subscripts.
   x[,4] # 4th column of matrix
   x[3,] # 3rd row of matrix
   x[2:4,1:3] # rows 2,3,4 of columns 1,2,3
```

Matrices

b. Useful matrix functions

***many of these will work on higher dimensional arrays too

-*dim()*: returns the dimensions (number of rows and columns) of the matrix

-*nrow()*: returns the number of rows in the matrix

-*ncol()*: returns the number of columns in the matrix

-*rownames()*: returns the row names of the matrix; can also be used for assignment

-*colnames()*: returns the column names of the matrix; can also be used for assignment

-*rbind()*: add a vector to a specified matrix as a new row at the bottom of the matrix

-*cbind()*: add a vector to a specified matrix as a new column at the furthest right

-*%*%*: matrix multiplication

-*t()*: transpose the matrix

-*colMeans()*: calculate the mean of each column of the matrix

-*colSums()*: calculate the sum of each column of the matrix

-*apply()*: applies a function that works on a vector to each row or column of a matrix

dimnames()

Element Access in Matrices

```
> M=matrix(1:4,nrow=2,ncol=2)
> M[2,1]
[1] 2
> M[2,2]
[1] 4
> M[4]
[1] 4
> M[,2]
[1] 3 4
```

Matrix operations

```
# matrix addition
matC = matA + matB

# matrix subtraction
matC = matA - matB

# scalar multiplication
matA = matrix(c(3,-1,0,5),2,2,byrow=TRUE)
matC = 2*matA

# matrix multiplication
matA = matrix(1:4,2,2,byrow=TRUE)
matB = matrix(c(1,2,1,3,4,2),2,3,byrow=TRUE)

matC = matA%*%matB
matC
```

Matrix operations

```
# create identity matrix
matI = diag(2)
matI
matA = matrix(c(1,2,3,4), 2, 2, byrow=TRUE)
matI%*%matA
matA%*%matI

# matrix inversion
matA
matA.inv = solve(matA)
matA.inv
matA%*%matA.inv
matA.inv%*%matA
```

Vector to matrix

- names(xvec) = c("x1", "x2", "x3")
- dim(xvec)

coerce vector to class matrix: note #column vector is created

- xvec = as.matrix(xvec)
- xvec
- class(xvec)

Data frames

Data frames

A data frame is more general than a matrix, in that different columns can have different modes (numeric, character, factor, etc.).

```
d <- c(1,2,3,4)
e <- c("red", "white", "red", NA)
f <- c(TRUE,TRUE,TRUE,FALSE)
mydata <- data.frame(d,e,f)
names(mydata) <- c("ID","Color","Passed")
   #variable names
```

Element access in Data frames

There are a variety of ways to identify the elements of a dataframe .

myframe[3:5] # columns 3,4,5 of dataframe
myframe[c("ID","Age")] # columns ID and Age from dataframe
myframe$X1 # variable x1 in the dataframe

Operations on Dataframes

b. Useful data frame functions

-*rbind()*: add a vector to a specified matrix as a new row at the bottom of the matrix

-*cbind()*: add a vector to a specified matrix as a new column at the furthest right

-*colMeans()*: calculate the mean of each column of the matrix

-*colSums()*: calculate the sum of each column of the matrix

-*apply()*: applies a function that works on a vector to each row or column of a matrix

-*merge()*: joins two data frames together using a shared column as an index

-*lapply()*: analogous to *apply()*, but operates on lists and returns a list

-*sapply()*: the same functionality as *lapply()*, but returns a matrix or vector

Factors

Factors

The factor stores the nominal values as a vector of integers in the range [1... k] (where k is the number of unique values in the nominal variable), and an internal vector of character strings (the original values) mapped to these integers.

Creating factors

```
# variable f1 with y and n values
        f1 = factor(levels=c("y","n"))


# variable gender with 20 "male" entries and
# 30 "female" entries
        gender = c(rep("male",20), rep("female", 30))
        gender = factor(gender)


# stores gender as 20 1s and 30 2s and associates
# 1=female, 2=male internally (alphabetically)
```

Lists

Lists

An ordered collection of objects (components). A list allows you to gather a variety of (possibly unrelated) objects under one name.

Lists

```
# example of a list with 4 components -
a string, a numeric vector, a matrix, and a scalar

w = list(name="Fred", mynumbers=a,
mymatrix=y, age=5.3)

# example of a list containing two lists
v = c(list1,list2)
```

Lists

Identify elements of a list using the [[]] convention.

mylist[[2]] # 2nd component of the list

Arrays

Arrays

Arrays are similar to matrices but can have more than two dimensions. See **help(array)** for details.