R DATA STRUCTURES

(DSC 101)

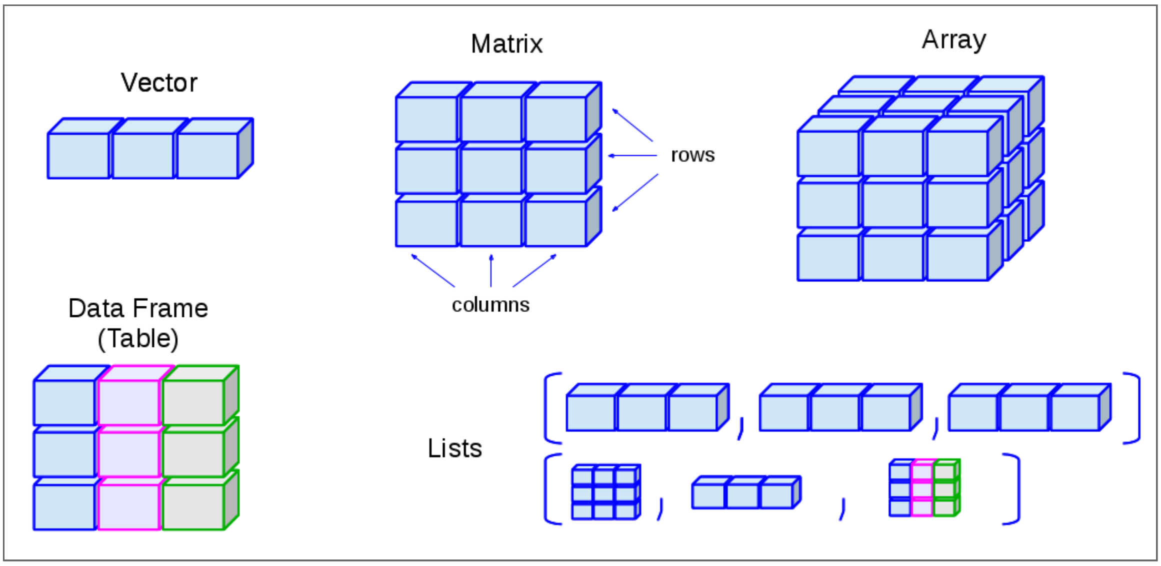
MATLOFF / BIRKENKRAHE

Created: 2021-09-20 Mo 19:43

WHAT WILL YOU LEARN?

- Preview of important data structures
- Vectors and scalars
- Character strings
- Matrices
- Lists
- Data Frames
- Classes
- Extended example

OVERVIEW



VECTORS AND SCALARS

VECTORS

- Storage modes: check `?mode`
- Functions: `mode`, `storage.mode`, `typeof`
- E.g. `numeric` (`double` or `integer`)
- Create a numeric vector of three elements!

```r
x <- c(1,2,3)  # integer
y <- rnorm(3)  # double
z <- 1:3       # integer

## print all three
x; y; z

## check mode
mode(x)
storage.mode(x)
typeof(x)

## check mode
mode(y)
storage.mode(y)
typeof(y)
```

SCALARS

- There are no scalars (numbers)
- Scalars are one-element vectors
- How could you show that?

```
s <- 1
s # prints vector of length 1

## change rownumber display
Nile[1:17]
options(width=100)
Nile[1:17]
```

CHARACTER STRINGS

- Single-element vectors of mode `character`
- Assign `x <- letters[1:3]` and print `x`
- Check the mode of `x`

```
x <- letters[1:3]
x
mode(x)
```

STRING MANIPULATION

- Create one numeric, two character vectors
- Concatenate character vectors with `paste`
- Split character vector with `strsplit`

```
## define vectors
x <- c(5,12,13)  # create numeric vect
x                # print x
length(x)        # print length of x
mode(x)          # print mode of x

y <- "abc"       # create character st
y
length(y)
mode(y)

z <- c("abc", "29 88")
z
length(z)
mode(z)
```

CONVERSION VS. COERCION

- `character` conversion: `as.character`
- `numeric` conversion: `as.numeric`
- Change numeric vector to character
- Change character vector to numeric

```
y # three real numbers
yc <- as.character(y)
yc
mode(yc)

x # three letters
xn <- as.numeric(x)
xn
mode(xn)
```

MATRICES

- A matrix is a rectangular array of numbers
- Matrices are vectors with rows and column attributes

CREATE MATRICES WITH `matrix`

- `matrix` creates a matrix from input values

```
A <- matrix()  # an empty 1 x 1 matrix
A
dim(A) # rows x columns

B <- matrix(NA) # an empty 1 x 1 matrix
B

C <- matrix(c(1,2)) # a 2 x 1 matrix
C
is.matrix(C)   # check if it's a matrix
```

ATTACHING ROWS AND COLUMNS

- `rbind` attaches rows
- `cbind` attaches columns

```
D <- rbind(c(1,4),c(2,2))
D

E <- cbind(c(1,4),c(2,2))
E
```

MATRIX ALGEBRA

- Matrices are multiplied with %*%

```
D %*% c(1,1)
E %*% c(1,1)
D %*% E
```

MATRIX INDEXING

- Matrices are indexed with two subscripts

```
D
D[1,2]  # row 1, col 2
D[,2]   # col 2
D[2,2]  # row 2, col 2
D[1,]   # row 1
```

LISTS

- Lists can contain different data types
- This is like a `struct` in C/C++
- Access elements with two-part names

```
x <- list(u=2, v="abc") # number and str
x
mode(x)

x$u # access list element u
x$v # access list element v

y <- paste(x$u,x$v)  # concatenation lea
y
mode(y)
length(y)
```

USE OF LISTS

- Combine multiple values
- Return list by function

```
hist(Nile)         # produces graph
hn <- hist(Nile) # save histogram as lis
mode(hn)           # mode of hn
print(hn)          # print hn (we can also
```

- More common way to show structure with `str`

```
str(hn)
```

DATA FRAMES

- Data frames are lists made of vectors
- Vectors can have different modes
- Data frames are rectangular but not matrices

CREATE DATA FRAME

- Turn a list into a data frame using `data.frame`

```
fam <- list(kids=c("Jack","Jill"), ages=
fam
d <- data.frame(fam)
d
```

- Turn vectors directly into a data frame

```
df <- data.frame(kids=c("Jack","Jill"),ages=c(12,10))
df
```

READ DATA FRAME FROM FILE

- Use `read.table` or `read.csv`
- You can read in straight from the web

```
## read csv without header information
pima_raw <-
  read.csv(file=
             "https://raw.githubusercont
           header=FALSE, sep=,)
head(pima_raw)
str(pima_raw)
```

- **[Download from Kaggle]** and read in from local machine

```
## read csv with header information
pima <- read.csv(file="/home/marcus/GitH
                 header=TRUE,
                 sep=,)
head(pima)
str(pima)
```

CLASSES

- R objects[1] are instances of *classes*
- Classes are *abstract* data types[2]
- Class instances are R lists with a class name

CLASS EXAMPLE: TIME SERIES

- The class of `Nile` is time series or `ts`

```
str(Nile)
class(Nile)
```

CLASS EXAMPLE: HISTOGRAM

- Non-graphical output of `hist()` has a class
- Compare also with `print(hn)`

```
hn <- hist(Nile)   # create a histogram c
mode(hn)           # the object is of moc
class(hn)          # its object class is
```

WHAT ARE CLASSES GOOD FOR?

- Classes are used by *generic* functions (**Chambers, 2002**)
- Generic = defines family of similar functions
- Each function fits a specific class
- This relates to R's package extensibility

GENERIC FUNCTION EXAMPLE: `summary()`

- Invoking `summary()` searches according to class, e.g.
  - Calling `summary()` on the output of `hist()`
  - Calling `summary()` on the output of `lm()` (regression)

```
summary(hn)
summary(lm(mtcars))
```

- You can call `plot()` on just about any R object, e.g.
  - Call `plot()` on a time series like `Nile`
  - Call `plot()` on a data frame like `mtcars`

```
plot(Nile)
plot(mtcars)
```

EXTENDED EXAMPLE: REGRESSION ANALYSIS

CONCEPT SUMMARY

CODE SUMMARY

CODE	DESCRIPTION

REFERENCES

Chambers J (2 Jan 2002). The Definition of Generic Functions and Methods [Website]. **Online: r-project.org.**