# Coding in R

Contents:


- Data Types
- Reading Data
- Sequence and Numbers
- Subsetting
- Control Structure
- Function
- Data and Time
- Loop Functions
- str Function
- Simulation
- Workspace


## Data Types

### R Objects and Attributes

- Atomic classes of objects
  - character
  - numeric
  - integer
  - complex
  - logical
- Most Basic Objects
  - vector: only contain objects of the same class
  - list: contain objects of different classes
- Numbers
  - usually treated as double precision real numbers
  - specify **L** suffix to get integer (i.e.`1L` gives integer 1)
  - special number: Inf, NaN
- Attributes
  - examples: names, dimnames, dimensions, class, length, user-defined metadata
  - accesss using `attributes()` function

### Vectors

- creating vectors
  - use `c()` function: `x<-c(1,2,3)`
  - use `vector()` function: `x<-vector("numeric",length=10)`
  - unique elements: `unique(c(1,2,2,3,4))`
- mixing objects
  - coercion such that each element is of the same class; nonsensical coercion results in NA
  - explicit coercion: `as.numeric(x)`, `as.logical(x)`, or `as.character(x)`
- character vector
  - join vectors: `paste(my_char,collapse=" ")`
  - join words: `paste(w1,w2,sep=" ")`
  - join multiple vectors: `paste(1:3,c("a","b","c"),sep="")` ### Lists
- creating lists
  - use `list()` function: `x<-list(1,"a",TRUE,1+4i)`

### Matrices

- creating matrix

  - initialize: `m<-matrix(nrow=2,ncol=3)`
  - construct column-wise: `m<-matrix(1:6, nrow=2, ncol=3)`
  - create from vector by adding dimension: `m<-1:10` and `dim(m)<-c(2,5)`
- attributes
  - `attributes(m)`
  - `dim(m)`
- binding vectors
  - column-binding: `cbind(x,y)`
  - row-binding: `rbind(x,y)`

**Factors**

- creating factors
  - default level: `x<-factor(c("yes","no","yes"))`
  - change order of level: `x<-factor(c("yes","no","yes"), levels=c("yes","no"))`
- display
  - `table(x)`
- unclass
  - `unclass(x)`

**Missing Values**

- properties
  - NA values have a class(i.e. integer NA, character NA, etc)
  - NaN value is also NA but the converse is not true
- testing
  - test NA: `is.na()`
  - test NaN: `is.nan()`
  - number of NA: `sum(is.na(data))`
  - `summary(data)` provides the count of NA's for each variable in the dataframe

**Data Frames**

- creating data frames
  - `read.table(file.txt)` or `read.csv(file.csv)`
  - `x<-data.frame(foo=1:4,bar=c(T,T,F,F))`
- attributes
  - `nrow(x)` and `ncol(x)`
  - column names: `colnames(my_frame)`, `colnames<-c(col1,col2)`
- read partial data
  - head: `head(data,row)`
  - tail: `tail(data,row)`

**Names**

- names for vectors
  - `x<-1:3` and `names(x)<-c("foo","bar","norf")`
  - `names(x)`
- names for lists
  - `x<-list(a=1, b=2, c=3)`
- names for matrices
  - `m<-matrix(1:4, nrow=2, ncol=2)` and `dimnames<-list(c("r1","r2"), c("c1","c2"))`

## Reading Data

### Reading Tabular Data

- reading data
  - tabular data: `read.table`, `read.csv`
  - reading lines: `readLines`
  - reading in R code files: `source`, `dget`
  - reading in saved workspace: `load`
- writing data
  - `write.table`
  - `writeLines`
  - `dump`
  - `dput`
  - `save`
- reading files with `read.table`
  - file or connection: `file`
  - if file has a header line: `header`
  - how column separated: `sep`
  - class of each column: `colClasses`
  - number of rows: `nrows`
  - comment character: `comment.char`
  - number of lines to skip: `skip`
  - if character will be coded as factors: `stringsAsFactors`

### Reading Large Table

- use `colClasses` argument
  - read some lines: `initial<-read.table("file.txt", nrows=100)`
  - find class: `classes<-sapply(initial,class)`
  - set colClasses value: `tabAll<-read.table("file.txt", colClasses=classes)`
- data size
  - `object.size(data)`
- set nrows
  - use Unix tool `wc` to count number of lines in the file
- compute memory requirement
- set `comment.char=""` if no commented lines

### Textual Data Formats

- dput and dget R objects
  - `y <- data.frame(a=1,b="a")` and `dput(y, file="y.R")`
  - `new.y <- dget("y.R")` and `new.y`
- dump R objects
  - `x <- "foo"` and `y<-data.frame(a=1, b="a")`
  - `dump(c("x","y"), file="data.R")`
  - `rm(x,y)` and `source("data.R")`

### Connections

- connection interfaces
  - file: `file`
  - compressed file with gzip: `gzfile`
  - compressed file with bzip2: `bzfile`
  - webpage: `url`
- file connections

  - check attributes: `str(file)`
  - same as `data <- read.csv("foo.txt")`: `con<-file("foo.txt","r")`, `data<-read.csv(con)`, and `close(con)`
- read lines
  - make connections: `con <- gzfile("words.gz")` or `con<-url("http://www.jhsph.edu","r")`
  - reading lines: `x <- readLines(con, 10)`

## Sequence and Numbers

### sequence

- operator `:`
  - `from:to`
- function `seq()`
  - `seq(from,to,by=0.1)`
  - `seq(from,to,length=10)`
  - `1:length(miser)` is same as `seq(along.with=myseq)` or `seq_along(myseq)`

### replicate

- function `rep()`
  - `rep(c(1,2),times=40)`
  - `rep(c(1,2),each=10)`

## Subsetting

### Operators

- `[` returns an object of the same class; can select multiple objects
- `[[` extracts elements of a list or a data frame; returns a single element
- `$` extracts elements of a list or data frame by names

### Vectors

- basic
  - `x[from:to]`
- logical statement
  - `x[condition1 & condition2]`
  - get indices: `which(x>10)`
  - boolean result: `any(x>0)`, `all(x>0)`
- random indexing
  - `x[c(idx1,idx2)]`
  - except some indices: `x[c(-idx1,-idx2)]` or `x[-c(idx1,idx2)]`
- names
  - `names(vect)`
  - assign names: `names(vet)<-c("name1","name2")`
- check identical
  - `identical(vect1,vect2)`

### Lists

- Basic
  - `x <- list(foo=1:4, bar=0.6)`
  - `x[1]`, `x[[1]]`, `x$bar`, `x[["bar"]]`, `x["bar"]`
- Select multiples objects
  - `x <- list(foo=1:4, bar=0.6, baz="hello")`

  - `x[c(1,3)]`
- Using computed indices with `[[`
  - `x <- list(foo=1:4, bar=0.6, baz="hello")`
  - `name <- "foo"` and `x[[name]]`
  - need `x$foo` not `x$name`
- `[[` can take an integer sequence
  - `x <- list(a=list(10,12,14), b=c(3.14, 2.8))`
  - `x[[c(1,3)]]` same as `x[[1]][[3]]` gives 14

### Matrix

- Use (i,j) type indices
  - `x <- matrix(1:6, 2, 3)`
  - element at (i,j): `x[2, 1]`
  - row or column: `x[1, ]` or `x[, 2]`
- Return elements as matrix
  - `x[1, 2, drop=FALSE]`
  - `x[1, , drop=FALSE]`

### Partial Matching

- Use `[[` or `$`
  - `x <- list(aardvark=1:5)`
  - `x$a`
  - `x[["a", exact=FALSE]]`

### Remove missing values

- Single objects
  - `x <- c(1, 2, NA, 4, NA, 5)`
  - `bad <- is.na(x)` and `x[!bad]`
- Multiple objects
  - `x <- c(1,2,NA,4,NA,5)` and `y <- c("a","b",NA,"d",NA,"f")`
  - `good <- complete.cases(x,y)` and `x[good]`, `y[good]`
- Data frame
  - `good <- complete.cases(airquality)`
  - `airquality[good, ][1:6, ]`

## Control Structure

### if-else

- basic
  - `if(condition){#something}else if(condition){#something}else{#smething}`
  - can assign whole structure to variable

### for-loops

- basic
  - `for(i in 1:10){ print(x[i]) }`
  - base on the length of x: `for(i in seq_along(x)) { print(x[i]) }`
  - `for(letter in x){ print(letter) }`
- nested for-loops
  - matrix: `for(i in seq_len(nrow(x))) for(j in seq_len(ncol(x)))`

**while-loops**

- basic
  - `while(condition){ #something }`
  - multiple conditions: `whilte(condition1 && condition2){ #something }`

**repeat, next, break**

- repeat
  - initiate infinite loop: `repeat{ if(condition){break} }`
  - must guarantee to stop, set hard limit on number of iterations using for-loop
- next
  - skip an iteration of a loop
  - `for(i in 1:10){ if(condition){skip} #something else}`
- break
  - `if(condition){break}`

## Function

**define function**

- creating functions
  - `f <- function(arguments){ #something }`
  - first class objects: treated as any R objects
  - function can be passed as arguments
  - function can be nested, can define function inside function
- default value
  - `f <- function(a, b=1, c=NULL)`

**arguments**

- function arguments
  - formal arguments are arguments in function definition
  - get list of formal arguments: `formals` function
  - arguments can be missing or use default value
- argument matching
  - matched positionally or by name
  - equivalent: `sd(mydata)`, `sd(x=mydata,na.rm=FALSE)`, `sd(na.rm=FALSE,mydata)`
  - not recommend messing orders
  - get arguments of function: `args(function)`
- partial argument matching
  - check for exact match for a named argument
  - check for a partial match
  - check for positional match
- lazy evaluation
  - only evaluated only as needed
- "..." argument
  - extend another function and dont want to copy the argument list
  - `myplot <- function(x,y,type="1",...){plot(x,y,type=type,...)}`
  - necessary when number of arguments not known in advance
  - example: `args(paste)` and `args(cat)`
- arguments after "..."
  - must be named explicitly and cannot be partially matched
  - `paste("a","b",sep=":")` not `paste("a","b",se=":")`

## Date and Time

### Dates in R

- date class
  - create date: `x <- as.Date("1970-01-01")`
  - unclass: `unclass(as.Date("1970-01-02"))`

### Times in R

- two classes
  - POSIXct: large integer, useful for data frame
  - POSIXlt: store a bunch of useful information
- POSIXlt usage
  - create time: `x <- Sys.time()`
  - create class: `p <- as.POSIXlt(x)`
  - get information: `names(unclass(p))` and `p$sec`
- POSIXct usage
  - create time: `x <- Sys.time()`
  - uncles to get secs: `unclass(x)`

### formatting times

- use strptime function
  - create date string: datestring <- c("January 10, 2012 10:40", "December 9, 2011 9:10")
  - format: `x <- strptime(datestring, "%B %d, %Y %H:%M")`

### operations on dates and times

- need same class to compare
- use `difftime()` function: `difftime(Sys.time(),t1,units="days")`
- use mathematical operations
- can change timezone

## Loop Functions

### lapply

- three arguments
  - a list x
  - a function or the name of a function FUN
  - other arguments via ...
- always return a list
  - `x <- list(a=1:5, b=rnorm(10))`
  - `lapply(x,mean)` with names preserved
  - apply to sequence: `x <- 1:4` and `lapply(x,runif)`
  - use ... arguments: `lapply(x,runif,min=0,max=10)`
- use of anonymous function
  - two matrices: `x <- list(a=matrix(1:4,2,2), b=matrix(1:6,3,2))`
  - extract first column: `lapply(x, function(let) elt[,1])`

### sapply

- simplify result of apply
  - if result is a list where every element is length 1, return a vector
  - if result is a list where every element of same length, return a matrix
  - if cannot figure out, return a list

**apply**

- used to evaluate a function over the margins of array
- arguments
  - array x
  - margin: integer vector indicating which margins should be retained
  - function to apply: FUN
  - ... arguments to be passed to FUN
- usage
  - `x <- matrix(rnorm(200),20,10)`
  - mean of each column: `apply(x,2,mean)` to preserve the columns and collapse rows
  - sum of each row: `apply(x,1,mean)`
  - shortcuts: `rowSums`, `rowMeans`, `colSums`, `colMeans`
- other ways to apply
  - quantiles: `apply(x,1,quantile,probs=c(0.25,0.75))`
  - multiple dimension: `a <- array(rnorm(2*2*10),c(2,2,10))`, `apply(a,c(1,2),mean)` is the same as `rowMeans(a,dims=2)`

**mapply**

- used to apply a function in parallel over a set of arguments
- arguments
  - FUN: function to apply
  - ...: arguments to apply over
  - MoreArgs: list of other arguments to FUN
  - SIMPLIFY: whether the result should be simplified
- usage
  - `mapply(rep,1:4,4:1)` is same as `list(rep(1,4),rep(2,3),rep(3,2),rep(4,1))`
  - instant vectorization: `noise <- fucntion(n,mean,sd)` and use `mapply(noise,1:5,1:5,2)` not `noise(1:5,1:5,2)`

**tapply**

- used to apply function over subsets of a vector
- arguments
  - vector x
  - INDEX: a factor or a list of factors
  - FUN: function to apply
  - ...: arguments for FUN
  - SIMPLIFY: result simplified
- usage
  - vector: `x <- c(rnorm(10),runif(10),rnorm(10,1))`
  - create groups: `f <- gl(3,10)`
  - apply function on x within group f: `tapply(x,f,mean)` or `tapply(x,f,range)`

**vapply**

- motivation
  - specify the format of result
- usage
  - single number: `vapply(data,function,numeric(1))`

**split**

- take a vector or other objects and split into groups determined by a factor or a list of factors
- arguments

  - x: a vector(or list) or data frame
  - f: a factor or a list of factors
  - drop: whether empty factor levels should be dropped
- usage
  - vector: `x <- c(rnorm(10),runif(10),rnorm(10,1))`
  - create groups: `f <- gl(3,10)`
  - split and use apply: `split(x,f)` and `lapply(split(x,f),mean)`
- split a data frame
  - look at partial data: `head(airquality)`
  - split by month: `s <- split(air quality,airquality$Month)`
  - compute mean: `lapply(s,function(x) colMeans(x[,c("Ozone","Solar.R","Wind")]))`
  - simplify results: `sapply(s,function(x) colMeans(x[,c("Ozone","Solar.R","Wind")],na.rm=TRUE))`
- split on more than one level
  - two levels: `x <- rnorm(10)`, `f1 <- gl(2,5)` and `f2 <- gl(5,2)`
  - combine levels: `interaction(f1,f2)`
  - drop empty levels: `str(split(x,list(f1,f2)), drop=TRUE)`

## str Function

### display structure in objects

- display class
- quick look at data

### usage

- function
  - `str(function)`
- vector
  - `x <- rnorm(100,2,4)`
  - `summary(x)` or `str(x)`
- data frame -`library(datasets)`
  - `head(airquality)` and `str(airquality)`
  - `str(split(airquality,airquality$Month))`
- matrix

## Simulation

### random numbers

- probability distributions
  - norm: random normal variables with given mean and sd
  - dnorm: evaluate normal probability density
  - pnorm: evaluate cumulative distribution function for normal distribution
  - ropes: random Poisson variates with a given rate
- four prefixes
  - d for density
  - r for random number generation
  - p for cumulative distribution
  - q for quantile function
- arguments
  - `dnorm(x,mean=0,sd=1,log=FALSE)`
  - `pnorm(q,mean=0,sd=1,lower.tail=TRUE,log.p=FALSE)`
  - `qnorm(p,mean=0,sd=1,lower.tail=TRUE,log.p=FALSE)`
  - `rnorm(n,mean=0,sd=1)`

- set seed
  - set seed to ensure reproducibility
  - use any seed number: `set.seed(1)`
  - always set seed when doing simulation
- repeat generation
  - `replicate(100,rpois(5,10))`

**generate from models**

- linear model
  - `set.seed(20)`
  - `x<-rnorm(100)` and `e<-rnorm(100,0,2)`
  - `y<-0.5+2*x+e` and `summary(y)`
  - `plot(x,y)`
- binary variable
  - `x<-rbinom(100,1,0.5)`
- generalized linear model
  - `x<-rnorm(100)` and `log.mu<-0.5+0.3*x`
  - `y<-rpois(100,exp(log.mu))`

**random sampling**

- draw randomly from a specified set of scalar objects
- usage
  - set seed `set.seed(1)`
  - without replacement: `sample(1:10,4)`
  - permutation `sample(1:10)`
  - with replacement: `sample(1:10,4,replace=TRUE)`
  - with probability: `sample(c(0,1),100,replace=TRUE,prob=c(0.3,0.7))`
- drawing from probability distribution with r* function

## Workspace

**directory**

- list files
  - `list.files()`
  - recursively: `list.files(recursive=TRUE)`
- directory operation
  - `getwd()` and `setwd(dir)`
  - `dir.create()`
  - create subdirectory: `dir.create("current/subdirectory",recursive=TRUE)`
  - delete: `unlink(dir,recursive=TRUE)`
- file
  - create: `file.create(filename)`
  - check existence: `file.exists(filename)`
  - get info: `file.info(filename)`
  - rename: `file.rename(from,to)`
  - copy: `file.copy(from,to)`
  - full path: `file.path(filename)`

**documentation**

- function: `args(function)`, `?function`
- operator: `?[backtick]operator[backtick]`