Lecture 2

Introduction to R

Assoc. Prof. Dr. Burkay Genç

26 Feb, 2024

R System Basics

Package system

- R is based on a powerful packaging system
- Almost everything is packaged as a separate library of functions
- Some packages are delivered by **default at installation**
  - base, boot, compiler, datasets, …
- Others, you have to install
  - dplyr, ggplot2, tidyr, …
- To install a package (i.e. dplyr):

```
install.packages("dplyr")
```

- To load an installed package:

```
library(dplyr)
```

Installed packages

- To get a list of all installed packages on the system

  `installed.packages()`

  or

  `library()`

- Check if a package can be updated

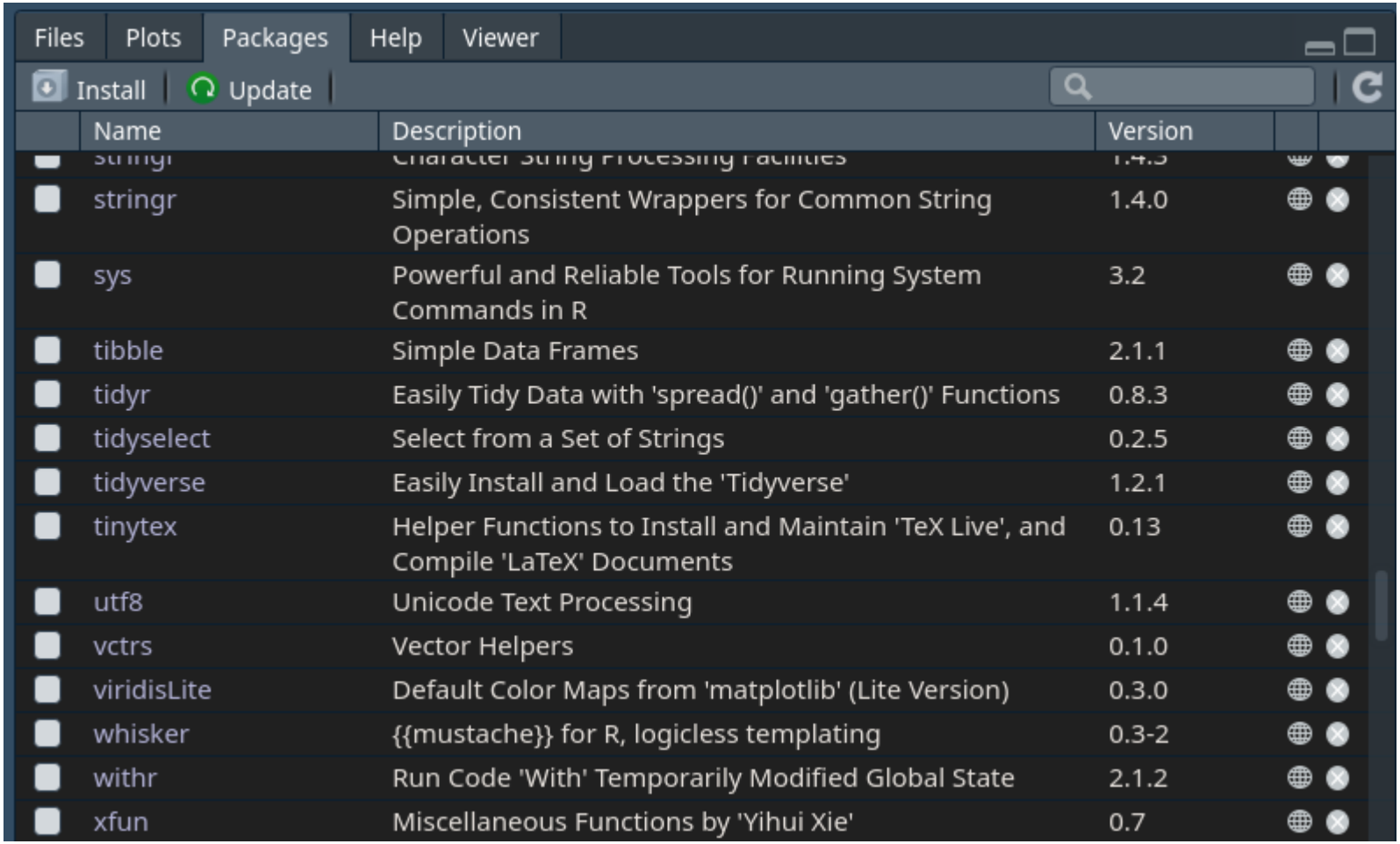
  `old.packages()`

- Update old packages

  `update.packages()`

- All of these can also be done from *RStudio* interface

RStudio Package Management



Setting a seed

You can set a seed to ensure reproducebility of results.

```
set.seed(1024)
runif(5, 0, 1)
```

```
## [1] 0.21808916 0.98763424 0.34846189 0.38104699 0.02098596
```

```
set.seed(1023)
runif(5, 0, 1)
```

```
## [1] 0.2493580 0.2529563 0.9496766 0.4891691 0.4063322
```

```
set.seed(1024)
runif(5, 0, 1)
```

```
## [1] 0.21808916 0.98763424 0.34846189 0.38104699 0.02098596
```

The same seed always produces the same random numbers.

This is quite essential in data studies as your results must be reproducible.

About random generator seeds

Note that these slides are generated with R.

Code scripts on each slide are being executed to produce these slides.

Whenever a seed is used on a slide, this will ensure that all results on a lecture are reproducible by you on your own PCs.

The seed setting in the previous slide also guarantees that the following slides are all reproducible.

Interaction with the R Console

```
4 + 3 / 5^2
```

```
## [1] 4.12
```

- The first line is the *input command*
- The second line is the *output*
- Notice the `[1]` at the beginning of the output
- In R, almost everything is a *vector* (array) of something
- Here, the output is a vector of numbers and the first item is `4.12`
- **Beware!!!** R vectors start with index 1

Interaction with the R Console

Producing 10 random normal values with a mean of 4 and standard deviation of 2:

```
rnorm(10, mean = 4, sd = 2)
```

```
##  [1] 5.347261 3.059237 4.444793 7.473732 3.772683 2.175777 6.118450 3.959706
##  [9] 3.423982 3.934415
```

Computing the mean of 5 randomly selected values between 1 and 10, both inclusive:

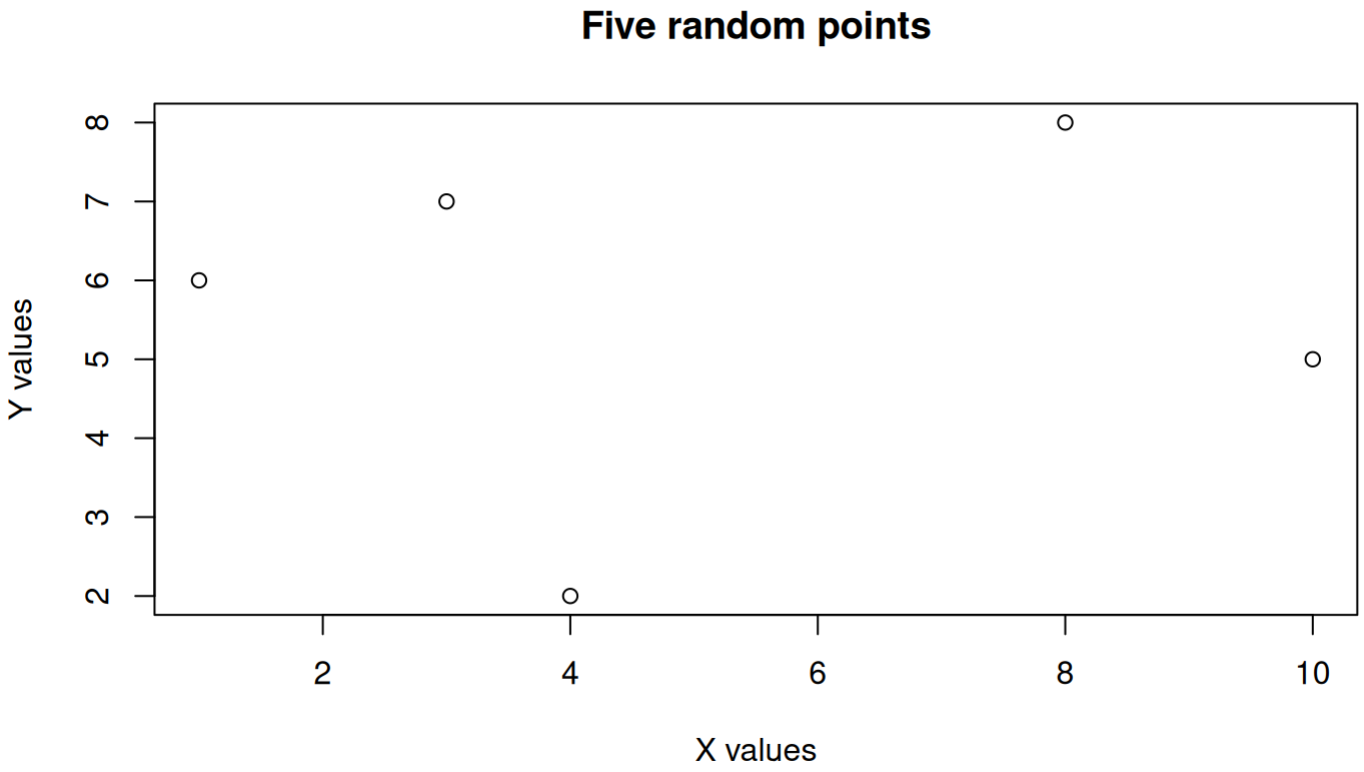
```
mean(sample(1:10, 5, replace = FALSE))
```

```
## [1] 6.6
```

Interaction with the R Console

Plots are drawn in a different window, (or tab in RStudio)

```
plot(x = sample(1:10,5), y = sample(1:10,5), main = "Five random points",
     xlab = "X values", ylab = "Y values")
```



Primitive Data Types

- *numeric*, *character*, *logical*
  - also, *integer* and *complex*
- Assign a value to a variable with `<-`

```
age <- 10
```

```
name <- "Ali"
```

```
registered <- FALSE
```

- You can re-assign on the fly

```
age <- 10
```

```
age <- "ten"
```

- Note that these are actually *vectors* of size 1.

Printing an Object's Value

- From the console just type the name of the object

```
age <- 10
age
```

```
## [1] 10
```

- But, be careful that this won't work in a script

```
age <- 10
age2 <- 20
age
age2
```

Printing an Object's Value

- Within a script, preferred method is

```
print(age)
```

```
## [1] 10
```

- You can also use `cat` to get rid of vector indices

```
cat(age)
```

```
## 10
```

Printing an Object's Value

- `print` and `cat` are different also at EOL (end of line)

```
x <- 5
y <- 10
print(x)
print(y)
```

```
## [1] 5
## [1] 10
```

```
cat(x)
cat(y)
```

```
## 510
```

So you must deliberately add an EOL when using cat

```
cat(x, "\n")
```

```
## 5
```

```
cat(y)
```

```
## 10
```

List and remove variables and objects

```
x <- 5
y <- 10
ls()
```

```
## [1] "x" "y"
```

```
rm(x)   # ... or, rm("x")
ls()
```

```
## [1] "y"
```

Functions

R Functions

- Functions are also objects

```
foo <- function()
{
   print("Hello World!")
}

foo()
```

```
## [1] "Hello World!"
```

R Functions

- Parameters can be provided

```
foo <- function(name)
{
   cat("Hello", name, "\n")
}

foo("Ali")
```

```
## Hello Ali
```

- Why not use `print()`?
  - `print` does not allow comma concatenation

R Functions

- Parameters can be initiated

```
foo <- function(name = "Hasan")
{
   cat("Hello", name, "\n")
}

foo()
```

```
## Hello Hasan
```

```
foo("Ali")
```

```
## Hello Ali
```

R Functions

- Parameters can be called by name

```
foo <- function(name = "Hasan", surname = "Kaya")
{
   cat("Hello", name, surname, "\n")
}

foo(surname = "Topal")
```

```
## Hello Hasan Topal
```

R Functions

- If called by name, order is not important

```
foo <- function(name = "Hasan", gender = "M")
{
   if (gender == "M")
      cat("Hello", "Mr.", name, "\n")
   else
      cat("Hello", "Ms.", name, "\n")
}

foo(name = "Ali", gender = "M")
```

```
## Hello Mr. Ali
```

```
foo(gender = "F", name = "Ayşe")
```

```
## Hello Ms. Ayşe
```

R Functions

- Some useful functions

```
max(1:10)
```

```
## [1] 10
```

```
min(1:10)
```

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

```
mean(1:10)
```

```
## [1] 5.5
```

R Functions

- Some useful functions

```
sd(1:10)
```

```
## [1] 3.02765
```

```
length(1:10)
```

```
## [1] 10
```

```
range(1:10)
```

```
## [1]  1 10
```

R Functions

- Is a function name already used?

```
exists("max")
```

```
## [1] TRUE
```

```
exists("burkay")
```

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

Vectors

Vectors

- A vector is a list of things of same type
- Create one with the `c()` function

```
names <- c("Ali", "Ayşe", "Burak")
names
```

```
## [1] "Ali"   "Ayşe"  "Burak"
```

```
ages <- c(12, 24, 36)
ages
```

```
## [1] 12 24 36
```

```
married <- c(F, T, T)
married
```

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

Vectors - Useful functions

```
length(names)
```

```
## [1] 3
```

```
mode(names)
```

```
## [1] "character"
```

Vectors - Coercion

```
ages <- c(12, 24, 36, "forty")
ages
```

```
## [1] "12"    "24"    "36"    "forty"
```

```
mode(ages)
```

```
## [1] "character"
```

```
mixed <- c(T, 12, "test")
mixed
```

```
## [1] "TRUE" "12"   "test"
```

Missing Values

- A missing value is represented by an `NA`

```
ages <- c(12, 24, 36, NA)
ages
```

```
## [1] 12 24 36 NA
```

- Be careful with computations if NA exists

```
mean(ages)
```

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

```
mean(ages, na.rm = T)
```

```
## [1] 24
```

Vector indexing

- Use brackets `vec[i]` to get element i of vec

```
ages
```

```
## [1] 12 24 36 NA
```

```
ages[1]
```

```
## [1] 12
```

```
ages[4]
```

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

Vector indexing

```
ages[4] <- 20
ages
```

```
## [1] 12 24 36 20
```

Subsetting

```
ages
```

```
## [1] 12 24 36 20
```

```
ages[1:3]
```

```
## [1] 12 24 36
```

```
ages[-1]
```

```
## [1] 24 36 20
```

```
ages[-1:-3]
```

```
## [1] 20
```

```
ages[-(1:3)]
```

```
## [1] 20
```

Subsetting

```
ages
```

```
## [1] 12 24 36 20
```

```
ages[c(1,3)]
```

```
## [1] 12 36
```

```
ages[c(-2,-4)]
```

```
## [1] 12 36
```

Boolean subsetting

A very frequently used technique is boolean subsetting. Items with `TRUE` indices are selected, while items with `FALSE` indices are discarded.

```
x <- 1:5
y <- c(F,T,T,F,T)
x[y]
```

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

This can be quite useful in many scenarios.

```
x <- 1:10
x[x > mean(x)]
```

```
## [1]  6  7  8  9 10
```

This works, because:

```
x > mean(x)
```

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

Batch assignments

```
ages <- 1:10
ages
```

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

```
ages[1:5] <- 20
ages
```

```
##  [1] 20 20 20 20 20  6  7  8  9 10
```

```
ages[3:5] <- c(99,98,97)
ages
```

```
##  [1] 20 20 99 98 97  6  7  8  9 10
```

```
ages[ages > 50] <- -1
ages
```

```
##  [1] 20 20 -1 -1 -1  6  7  8  9 10
```

Dynamic resizing

```
ages <- c(12, 24, 36)
ages
```

```
## [1] 12 24 36
```

```
ages[4] <- 48
ages
```

```
## [1] 12 24 36 48
```

```
ages[6] <- 72
ages
```

```
## [1] 12 24 36 48 NA 72
```

```
ages[is.na(ages)] <- mean(ages, na.rm = TRUE)
ages
```

```
## [1] 12.0 24.0 36.0 48.0 38.4 72.0
```

An automatic coercion to real numbers have occured.

Empty vectors

```
ages <- vector()
ages
```

```
## logical(0)
```

```
ages[1] <- 24
ages
```

```
## [1] 24
```

Empty vectors

You can create vectors of any size, but they will be initialized by default values:

```
ages <- numeric()
ages
```

```
## numeric(0)
```

```
ages <- numeric(10)
ages
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0
```

```
gender <- character(10)
gender
```

```
##  [1] "" "" "" "" "" "" "" "" "" ""
```

```
married <- logical(10)
married
```

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

Vector concatenation

You can concatenate vectors with `c()`:

```
x <- c(1, 2, 3)
x <- c(x, 4)
x
```

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

```
x <- c(x, x)
x
```

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

Vectorization

- R supports many operations and functions to work on vectors
- Scalars will be automatically converted to vectors
  - Below `1` is treated as `[1,1,1,1,1,1,1,1,1,1]`

```
x <- 1:10
y <- x + 1
y
```

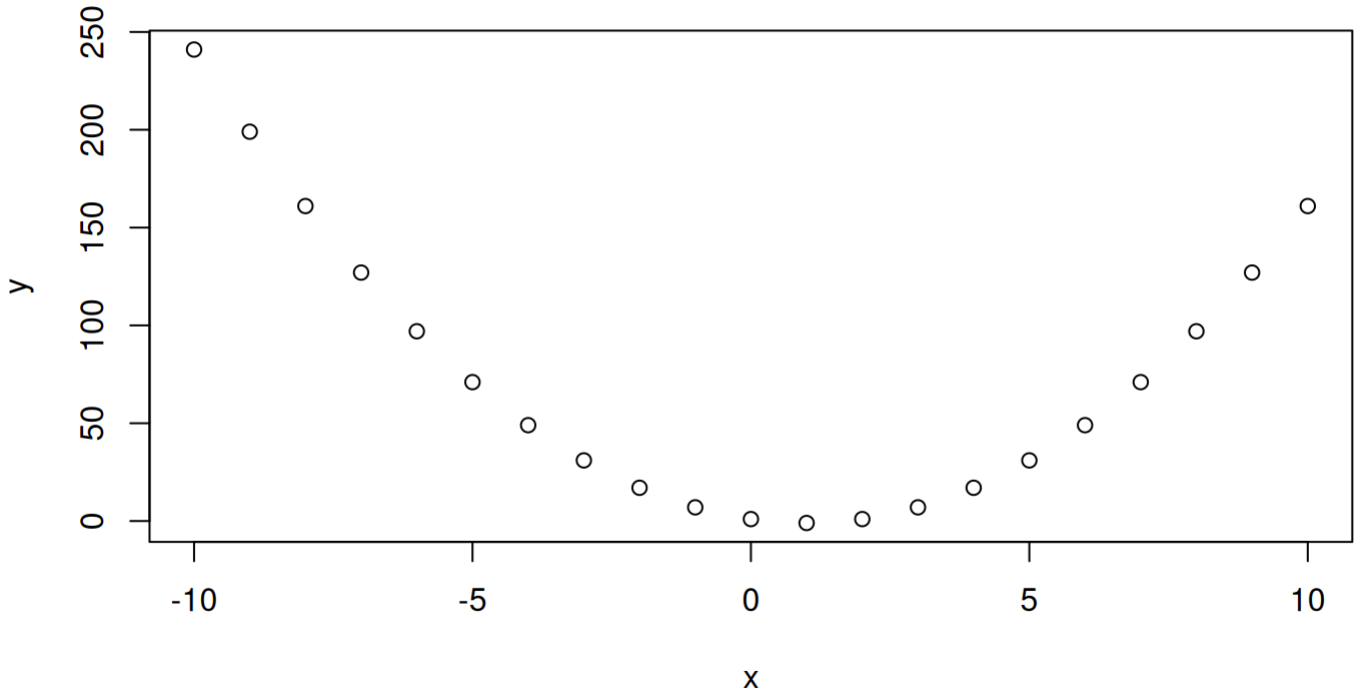
```
##  [1]  2  3  4  5  6  7  8  9 10 11
```

```
y <- x^2
y
```

```
##  [1]   1   4   9  16  25  36  49  64  81 100
```

Vectorization

```
x <- -10:10
y <- 2*x^2 - 4*x + 1
plot(x, y)
```



Vectorization - Recycling

```
x <- 1:10
y <- 10:1
x + y
```

```
##  [1] 11 11 11 11 11 11 11 11 11 11
```

```
y <- 1:5
x
x + y
```

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

```
y <- 1:3
x
x + y
```

```
## Warning in x + y: longer object length is not a multiple of shorter object
## length
```

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

Factors

Factors

```
color <- c("red", "blue", "blue", "blue", "red")
color
```

```
## [1] "red"  "blue" "blue" "blue" "red"
```

```
color <- factor(color)
color
```

```
## [1] red  blue blue blue red
## Levels: blue red
```

```
color[3]
```

```
## [1] blue
## Levels: blue red
```

```
color[4] == "blue"
```

```
## [1] TRUE
```

Factors

```
color
```

```
## [1] red  blue blue blue red
## Levels: blue red
```

```
color[4] <- "red"
color
```

```
## [1] red  blue blue red  red
## Levels: blue red
```

```
color[4] <- "green"
```

```
## Warning in `[<-.factor`(`*tmp*`, 4, value = "green"): invalid factor level, NA
## generated
```

```
color
```

```
## [1] red  blue blue <NA> red
## Levels: blue red
```

Factors - Forced levels

```
color <- factor(c("red", "blue", "blue", "blue", "red"),
                levels = c("blue", "red", "green"))
color
```

```
## [1] red  blue blue blue red
## Levels: blue red green
```

```
color[4] <- "green"
color
```

```
## [1] red   blue  blue  green red
## Levels: blue red green
```

Factors - Useful functions

```
levels(color)
```

```
## [1] "blue"  "red"   "green"
```

```
length(color)
```

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

```
table(color)
```

```
## color
##  blue   red green
##     2     2     1
```

Cross tabulation

```
hair <- factor(c("Red", "Brown", "Brown", "Black", "Yellow", "Yellow", "Black", "Brown"))
eye <- factor(c("Brown", "Black", "Brown", "Green", "Black", "Brown", "Green", "Black"))
table(hair, eye)
```

```
##         eye
## hair     Black Brown Green
##   Black      0     0     2
##   Brown      2     1     0
##   Red        0     1     0
##   Yellow     1     1     0
```

Cross tabulation

```
t <- table(hair, eye)
margin.table(t, 1)
```

```
## hair
##  Black  Brown    Red Yellow
##      2      3      1      2
```

```
margin.table(t, 2)
```

```
## eye
## Black Brown Green
##     3     3     2
```

Cross tabulation

```
t <- table(hair, eye)
prop.table(t, 1)
```

```
##         eye
## hair          Black     Brown     Green
##   Black  0.0000000 0.0000000 1.0000000
##   Brown  0.6666667 0.3333333 0.0000000
##   Red    0.0000000 1.0000000 0.0000000
##   Yellow 0.5000000 0.5000000 0.0000000
```

```
prop.table(t, 2)
```

```
##         eye
## hair          Black     Brown     Green
##   Black  0.0000000 0.0000000 1.0000000
##   Brown  0.6666667 0.3333333 0.0000000
##   Red    0.0000000 0.3333333 0.0000000
##   Yellow 0.3333333 0.3333333 0.0000000
```

```
prop.table(t)
```

```
##         eye
## hair     Black Brown Green
##   Black  0.000 0.000 0.250
##   Brown  0.250 0.125 0.000
##   Red    0.000 0.125 0.000
##   Yellow 0.125 0.125 0.000
```

Sequences

Sequences

```
seq(1, 30, 1)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30
```

```
seq(1, 30, 5)
```

```
## [1]  1  6 11 16 21 26
```

```
seq(-1, 1, 0.1)
```

```
##  [1] -1.0 -0.9 -0.8 -0.7 -0.6 -0.5 -0.4 -0.3 -0.2 -0.1  0.0  0.1  0.2  0.3  0.4
## [16]  0.5  0.6  0.7  0.8  0.9  1.0
```

```
seq(1, -1, -0.2)
```

```
##  [1]  1.0  0.8  0.6  0.4  0.2  0.0 -0.2 -0.4 -0.6 -0.8 -1.0
```

Sequences

```
seq(1, 30, length=6)
```

```
## [1]  1.0  6.8 12.6 18.4 24.2 30.0
```

```
seq(from = 10, length = 5, by = 3)
```

```
## [1] 10 13 16 19 22
```

Repetitive sequences

```
rep(1, 10)
```

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

```
rep(1:2, 5)
```

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

```
rep(1:2, each = 5)
```

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

Useful Tricks

Generating factors

```
gl(3, 5)
```

```
##  [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
## Levels: 1 2 3
```

```
gl(3, 5, labels = c("Red", "Green", "Blue"))
```

```
##  [1] Red   Red   Red   Red   Red   Green Green Green Green Green Blue  Blue
## [13] Blue  Blue  Blue
## Levels: Red Green Blue
```

Other generators

```
rnorm(10, 1, 2)
```

```
##  [1]  4.21060332  3.89519151  0.74339555 -0.07785289  1.78317210  2.75843405
##  [7] -0.64946418  2.46575285 -0.32982902  1.72177110
```

```
rpois(10, 3)
```

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

```
rbinom(10, 3, 0.3)
```

```
##  [1] 1 2 0 2 0 0 1 0 2 0
```

```
runif(10, -10, 10)
```

```
##  [1] -4.968684 -3.225994 -7.735908  5.628258  6.942104  3.860872  4.541266
##  [8] -2.912779 -4.304087 -6.685231
```

More subsetting

```
x <- rpois(10, 3)
x
```

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

```
x > 3
```

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

```
x[x > 3]
```

```
## [1] 5 4 4
```

```
which(x > 3)
```

```
## [1] 6 7 8
```

More subsetting

```
x <- rpois(10, 3)
x
```

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

```
x > 3
```

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

```
x[x > 3]
```

```
## [1] 4 5 5 4
```

```
which(x > 3)
```

```
## [1]  3  5  8 10
```

More subsetting

```
x
```

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

```
x[x > 3] <- 3
x
```

```
##  [1] 2 1 3 3 3 1 3 3 1 3
```

Binary comparison

```
x <- c(F, F, T, T)
y <- c(F, T, T, F)
x & y
```

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

```
x | y
```

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

Binary comparison

As of 4.3.0, `&&` and `||` only works with vectors of length 1

```
x <- T
y <- F
x && y
```

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

```
x || y
```

```
## [1] TRUE
```

```
x & y
```

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

```
x | y
```

```
## [1] TRUE
```

So, simply use `&` or `|`.

Named indexes

```
ages <- c(12, 24, 36)
names(ages) <- c("Ali", "Ayşe", "Burak")
ages
```

```
##   Ali  Ayşe Burak
##    12    24    36
```

```
ages["Burak"]
```

```
## Burak
##    36
```

```
ages[c("Ali", "Burak")]
```

```
##   Ali Burak
##    12    36
```

Named indexes

```
ages <- c(Ali = 12, Ayşe = 24, Burak = 36)
ages
```

```
##   Ali  Ayşe Burak
##    12    24    36
```

Matrices and Arrays

Matrices

```
m <- c(1, 2, 3, 4, 5, 6, 7, 8)
dim(m)
```

```
## NULL
```

```
dim(m) <- c(2, 4)
m
```

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

```
m <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8), 2, 4)
m
```

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

Matrices - By row

```
m <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8), 2, 4, byrow = T)
m
```

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

Matrices - Indexing

```
m
```

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

```
m[2, 3]
```

```
## [1] 7
```

```
m[2,]
```

```
## [1] 5 6 7 8
```

```
m[,3]
```

```
## [1] 3 7
```

Matrices - Indexing

```
m
```

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

```
m[2, -2]
```

```
## [1] 5 7 8
```

```
m[1:2, 2:3]
```

```
##      [,1] [,2]
## [1,]    2    3
## [2,]    6    7
```

Matrices - Forced matrix output

```
m[2,]
```

```
## [1] 5 6 7 8
```

```
m[2, , drop = F]
```

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

Matrices - Binding

```
m <- matrix(1:9, 3, 3)
v <- c(10, 11, 12)
rbind(m, v)
```

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

```
cbind(m, v)
```

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

Matrices - Naming

```
m <- matrix(1:9, 3, 3)
colnames(m) <- c("A", "B", "C")
rownames(m) <- c("x", "y", "z")
m
```

```
##   A B C
## x 1 4 7
## y 2 5 8
## z 3 6 9
```

```
m["y", "C"]
```

```
## [1] 8
```

Arrays

- Arrays are matrices with higher dimensions
  - Don’t confuse with C and Java type arrays

```
a <- array(1:24, dim = c(4, 3, 2))
a
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]   13   17   21
## [2,]   14   18   22
## [3,]   15   19   23
## [4,]   16   20   24
```

Recycling in Matrices and Arrays

```
m <- matrix(1, 3, 3)
m
```

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

```
m <- matrix(1:3, 3, 3)
m
```

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

Matrix operations

```
m1 <- matrix(1:9, 3, 3)
m2 <- matrix(2, 3, 3)
m1
```

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

```
m2
```

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

```
m1 + m2
```

```
##      [,1] [,2] [,3]
## [1,]    3    6    9
## [2,]    4    7   10
## [3,]    5    8   11
```

```
m1 * m2
```

```
##      [,1] [,2] [,3]
## [1,]    2    8   14
## [2,]    4   10   16
## [3,]    6   12   18
```

```
m1 %*% m2  # Matrix multiplication
```

```
##      [,1] [,2] [,3]
## [1,]   24   24   24
## [2,]   30   30   30
## [3,]   36   36   36
```

Lists

Lists

- Lists allow you to package different types of objects

```
my.lst <- list(stud.id=34453,
               stud.name="Ali",
               stud.marks=c(14.3,12,15,19))
my.lst
```

```
## $stud.id
## [1] 34453
##
## $stud.name
## [1] "Ali"
##
## $stud.marks
## [1] 14.3 12.0 15.0 19.0
```

Lists

```
my.lst
```

```
## $stud.id
## [1] 34453
##
## $stud.name
## [1] "Ali"
##
## $stud.marks
## [1] 14.3 12.0 15.0 19.0
```

```
my.lst[1]
```

```
## $stud.id
## [1] 34453
```

```
my.lst[[1]]
```

```
## [1] 34453
```

Lists

```
my.lst$stud.id          # same as my.lst[[1]]
```

```
## [1] 34453
```

```
my.lst["stud.id"]       # same as my.lst[1]
```

```
## $stud.id
## [1] 34453
```

```
my.lst[["stud.id"]]     # same as my.lst[[1]]
```

```
## [1] 34453
```

Lists - tricks with names

```
l <- list(names = c("Ali", "Ayşe", "Burak"),
          ages = c(12, 24, 36),
          registered = c(T, F, F))
l$names
```

```
## [1] "Ali"   "Ayşe"  "Burak"
```

```
l$nam
```

```
## [1] "Ali"   "Ayşe"  "Burak"
```

```
l$r
```

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

Lists - tricks with names

```
l <- list(names = c("Ali", "Ayşe", "Burak"),
          ages = c(12, 24, 36),
          agents = c(2, 1, 2),
          registered = c(T, F, F))
l$age
```

```
## NULL
```

```
l$agen
```

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

Lists - tricks with names

```
l <- list(names = c("Ali", "Ayşe", "Burak"),
          ages = c(12, 24, 36),
          registered = c(T, F, F))
names(l)
```

```
## [1] "names"      "ages"       "registered"
```

```
names(l)[2] <- "age"
l
```

```
## $names
## [1] "Ali"   "Ayşe"  "Burak"
##
## $age
## [1] 12 24 36
##
## $registered
## [1]  TRUE FALSE FALSE
```

Lists - adding to a list

```
l <- list(names = c("Ali", "Ayşe", "Burak"),
          ages = c(12, 24, 36),
          registered = c(T, F, F))
l$agents <- c(2, 1, 2)
l
```

```
## $names
## [1] "Ali"   "Ayşe"  "Burak"
##
## $ages
## [1] 12 24 36
##
## $registered
## [1]  TRUE FALSE FALSE
##
## $agents
## [1] 2 1 2
```

Lists - removing from a list

```
l <- list(names = c("Ali", "Ayşe", "Burak"),
          ages = c(12, 24, 36),
          registered = c(T, F, F))
l$ages <- NULL
l
```

```
## $names
## [1] "Ali"   "Ayşe"  "Burak"
##
## $registered
## [1]  TRUE FALSE FALSE
```

Lists - removing from a list

```
l <- list(names = c("Ali", "Ayşe", "Burak"),
          ages = c(12, 24, 36),
          registered = c(T, F, F))
l <- l[-2]
l
```

```
## $names
## [1] "Ali"   "Ayşe"  "Burak"
##
## $registered
## [1]  TRUE FALSE FALSE
```

Lists - concatenation

```
l1 <- list(1, "a", T)
l2 <- list("red", 1.72)
l <- c(l1, l2)
l
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] "red"
##
## [[5]]
## [1] 1.72
```