Introduction to R

Business Analytics

Stefan Feuerriegel

Today's Lecture

Objectives

1. Being able to perform simple calculations in R
2. Understanding the concepts of variables
3. Handling vectors and matrices

Outline

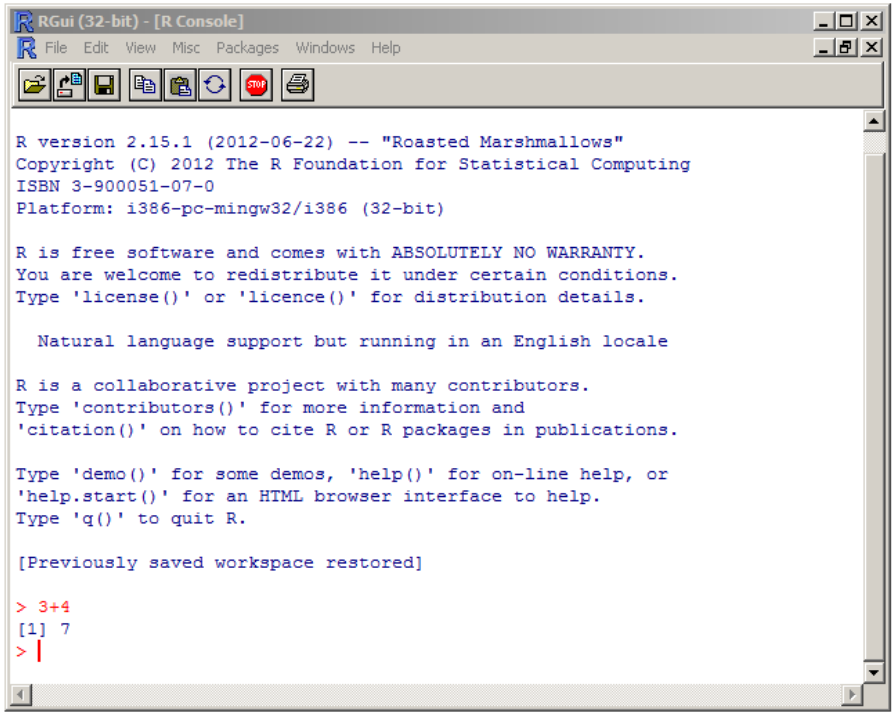
Outline

Examples of Statistical Software

Excel	Limited capabilities for statistics; good for data preprocessing
SPSS	Easy/good for standard procedures
SAS	Good for large data sets and complicated analysis
STATA	Common in research; various estimators and statistical tests
EViews	Strong focus on time series analysis
Matlab	Mathematical programming, but statistical methods limited

What is R?

- Free software environment aimed at statistical computing
- Supports many operating systems (Linux, Mac OS X, Windows)
- Very frequently used in psychology, bioinformatics, statistics, econometrics, and machine learning

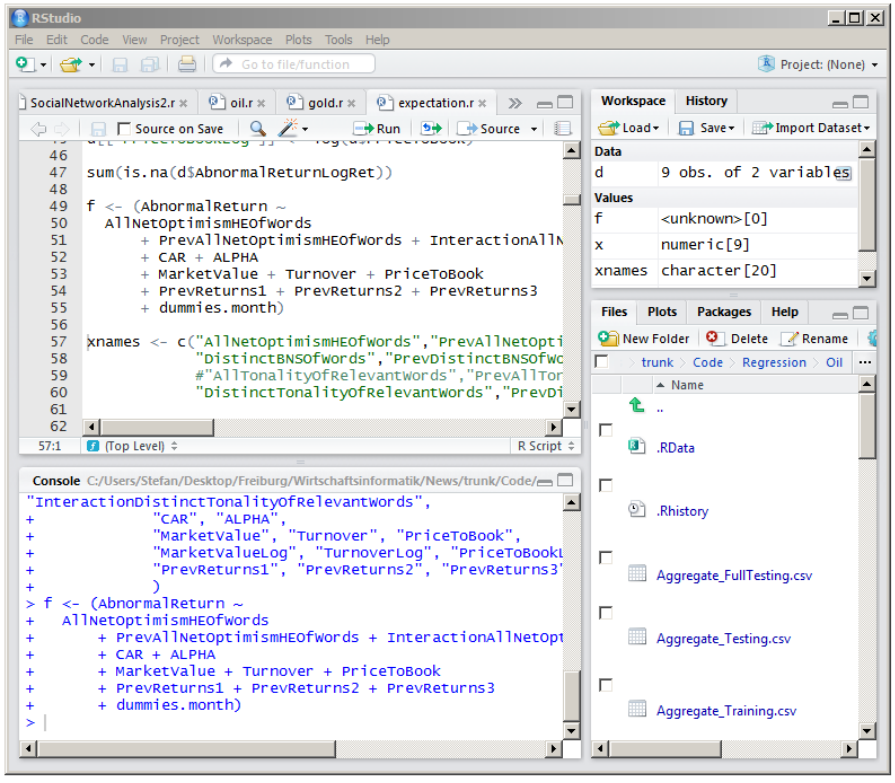


Retrieving R

Download at `http://www.r-project.org`

R Studio as Editor

- Instead of typing commands into the R Console, you can generate commands by an editor and then send them to the R window
- . . . and later modify (correct) them and send again



Retrieving R Studio (recommended)

Download at `http://www.rstudio.com/`

Outline

First Example

```
3*(4+2)

## [1] 18
```

Arithmetic Operations

```
1+2*3
```

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

```
3/4+2
```

```
## [1] 2.75
```

```
2*pi-pi
```

```
## [1] 3.141593
```

```
0/0
```

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

Operation	Description	Example	Result
`+`	Plus	`3+4`	7
`-`	Minus	`3-4`	-1
`*`	Times	`3*4`	12
`/`	Divide	`3/4`	0.75
`^`	Exponentiation	`3^4`	$3^4 = 81$

Logic Operators

Comparison Operators

Operators `<`, `<=`, `==`, `!=`, `>=`, `>` return boolean values `TRUE` or `FALSE`

```
3 < 4
```

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

```
3 > 4
```

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

```
3 <= 4
```

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

```
4 == 4
```

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

```
3 != 4
```

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

Brackets, Comments and Decimal Points

- Brackets can be used to prioritize evaluations

```
3*(4+2)
```

```
## [1] 18
```

- Important to use a point instead of a comma!

```
3.141
```

```
## [1] 3.141
```

- Comments via #

```
3+4 # will be ignored
```

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

Mathematical Functions

- Square root

```
sqrt(1+1)
```

```
## [1] 1.414214
```

- Logarithm to the base 10

```
log10(10*10*10)
```

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

- Sinus function and rounding

```
sin(pi) # rarely exact: R uses limited number of digits
```

```
## [1] 1.224606e-16
```

```
round(sin(pi))
```

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

Mathematical Functions

Function	Description	Example	Result
`abs()`	Absolute Value	`abs(3-4)`	$+1$
`round()`	Rounding	`round(3.14)`	≈ 3
`sqrt()`	Square Root	`sqrt(81)`	$\sqrt{81} = 9$
`sin()`	Sine	`sin(0)`	$\sin 0 = 0$
`cos()`	Cosine	`cos(0)`	$\cos 0 = 1$
`tan()`	Tangent	`tan(0)`	$\tan 0 = 0$
`log()`	Natural Logarithm	`log(e)`	$\ln e = 1$
`log10()`	Common Logarithm	`log10(100)`	$\log_{10} 100 = 2$

Exercise: Mathematical Functions

Question

- What is the value of `abs(3-4*5)`?

```
abs(3-4*5)

## [1] 17
```

Variables

```
x <- 2
x

## [1] 2

x+3

## [1] 5

x

## [1] 2

x <- x+4
x

## [1] 6
```

- Variables store values during a session
- Value on right is assigned to variable preceding "<−"
- No default output after assignment
- Recommended names consist of letters A–Z plus "_" and "."
- Must not contain minus!
  - Should be different from function names, e. g. `sin`
  - Good: `x`, `fit`, `ratio`, etc.
- Warning: naming is case-sensitive
  - i. e. `x` and `X` are different

Exercise: Variables

Question

- What is the value of `z`?

```
x <- 2
x <- x+1
y <- 4
z <- x+y
x <- x+1
z <- z+x
```

```
z
```

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

Exercise: Variables

Question

- What is the value of `z`?

```
x <- 2
x <- x+1
y <- 4
z <- x+y
x <- x+1
z <- z+x
```

```
z
```

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

Strings

- Sequence of characters are named strings
- Surrounded by double quotes (")
- Necessary for e. g. naming column names

```
"Text"
```

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

```
"3.14"
```

```
## [1] "3.14"
```

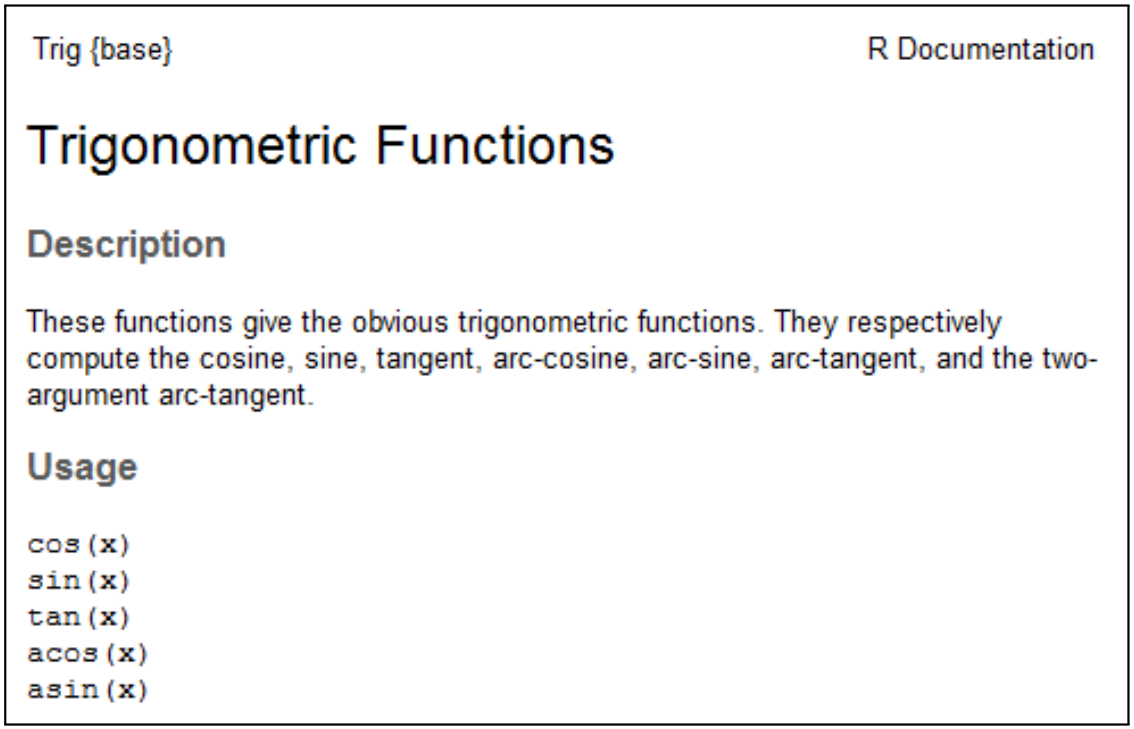
```
"3.14"+1 # mixing strings and numbers does not work
```

```
## Error in "3.14" + 1:  non-numeric argument to binary
operator
```

Help Pages

Accessing help pages for each function via `help(func)`

```
help(sin)
```



Trig {base} R Documentation

Trigonometric Functions

Description

These functions give the obvious trigonometric functions. They respectively compute the cosine, sine, tangent, arc-cosine, arc-sine, arc-tangent, and the two-argument arc-tangent.

Usage

```
cos(x)
sin(x)
tan(x)
acos(x)
asin(x)
```

Outline

Creating and Accessing Vectors

- Create vector filled with zeros via `numeric(n)`

```
numeric(4)

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

- Vector elements are concatenated via `c(...)`

```
x <- c(4, 0, 6)
x

## [1] 4 0 6
```

- Accessing individual elements via squared brackets `[]`

```
x[1] # first component

## [1] 4
```

- Selecting a range of elements

```
x[c(2,3)]

## [1] 0 6
```

- Selecting everything but a subset of elements

```
x[-1]

## [1] 0 6

x[-c(2,3)]

## [1] 4
```

- Dimension via `length()`

```
length(x)

## [1] 3
```

Updating Vectors

```
x <- c(4, 0, 6)
```

- Replacing values

```
x[1] <- 2 # replace first component
x
```

```
## [1] 2 0 6
```

- Appending elements

```
y <- c(x, 8) # append an element
y
```

```
## [1] 2 0 6 8
```

Vectors: Concatenation

```
x <- c(4, 0, 6)
y <- c(8, 9)
```

- Combining several vectors is named concatenation

```
z <- c(x, y) # concatenating two vectors
z

## [1] 4 0 6 8 9
```

- Replicating elements by `rep(val, count)` to form vectors

```
rep(1, 5) # 5-fold replication of the value 1

## [1] 1 1 1 1 1

rep(c(1, 2), 3) # repeat vector 3 times

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

Vector Functions

```
x <- c(1, 2, 3, 0, 10)
```

- Average value

```
mean(x)
```

```
## [1] 3.2
```

- Variance

```
var(x)
```

```
## [1] 15.7
```

- Sum of all elements

```
sum(x)
```

```
## [1] 16
```

Exercise: Vectors

Question

- How to compute a standard deviation of $x = \begin{bmatrix} 1 \\ 4 \\ 9 \end{bmatrix}$?
  - `sqr(var(x))`
  - `sqrt(var(x))`
  - `sd(x)`

```
x <- c(1, 4, 9)
```

Solution A

```
sqrt(var(x))
```

```
## [1] 4.041452
```

Solution B

```
sd(x)
```

```
## [1] 4.041452
```

Exercise: Vectors

Question

- How to compute a standard deviation of $x = \begin{bmatrix} 1 \\ 4 \\ 9 \end{bmatrix}$?
  - `sqr(var(x))`
  - `sqrt(var(x))`
  - `sd(x)`

```
x <- c(1, 4, 9)
```

Solution A

```
sqrt(var(x))

## [1] 4.041452
```

Solution B

```
sd(x)

## [1] 4.041452
```

Vector Operations

```
x <- c(1, 2)
y <- c(5, 6)
```

- Scaling

```
10*x
```

```
## [1] 10 20
```

- Addition

```
x+y
```

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

```
10+x
```

```
## [1] 11 12
```

- Be careful with functions such as `sin()` on vectors!

Generating Sequences

- Integer sequences

```
1:4
```

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

```
4:1
```

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

- Arbitrary sequences

```
(1:10)/10
```

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

```
seq(4, 5, 0.1) # notation: start, end, step size
```

```
##  [1] 4.0 4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 4.9 5.0
```

Exercise: Vectors

Question

- How to compute $\sum_{i=1}^{100} i$?
  - `sum(1:100)`
  - `sum(1,100)`
  - `sum(1-100)`

```
sum(1:100)

## [1] 5050
```

Exercise: Vectors

Question

- How to compute $\sum_{i=1}^{100} i$?
  - `sum(1:100)`
  - `sum(1,100)`
  - `sum(1-100)`

```
sum(1:100)

## [1] 5050
```

Outline

Matrices from Combining Vectors

- Generating matrices by combining vectors with `cbind(...)`

```
height <- c(163, 186, 172)
shoe_size <- c(39, 44, 41)
m <- as.data.frame(cbind(height, shoe_size))
```

  ...but exhausting!

- `as.data.frame(...)` necessary to store data of different types (numeric, strings, etc.)

Files formatted as Comma Separated Values

- Support of naive Excel format is unsatisfactory
- Recommended: Export as Comma Separated Values (CSV)
- In Excel via Save As $\rightarrow$ file type is CSV (Comma separated)
- Then: right mouse click $\rightarrow$ Open with $\rightarrow$ Text Editor $\rightarrow$ Check if there are commas

Example File: persons.csv

```
name,height,shoesize,age
Julia,163,39,24
Robin,186,44,26
Kevin,172,41,21
Max,184,43,22
Jerry,193,45,31
```

Matrices from Text Files

`read.csv(filename, ...)` imports data frame from text file

- `header=TRUE` specifies whether columns have names
- `sep=","` specifies column delimiter
- `as.data.frame(...)` guarantees output as data frame

```
d <- as.data.frame(read.csv("persons.csv",
     header=TRUE, sep=","))
d

##    name height shoesize age
## 1 Julia    163       39  24
## 2 Robin    186       44  26
## 3 Kevin    172       41  21
## 4   Max    184       43  22
## 5 Jerry    193       45  31
```

- Alternatively, choose path to file via `file.choose()` manually

```
d <- as.data.frame(read.csv(file.choose(),
     header=TRUE, sep=","))
```

Output: Matrices

- Show first 6 rows only (useful for large files)

```
head(d)
```

```
##    name height shoesize age
## 1 Julia    163       39  24
## 2 Robin    186       44  26
## 3 Kevin    172       41  21
## 4   Max    184       43  22
## 5 Jerry    193       45  31
```

- Show column names

```
str(d)
```

```
## 'data.frame': 5 obs. of  4 variables:
##  $ name    : Factor w/ 5 levels "Jerry","Julia",..: 2 5 3 4 1
##  $ height  : int  163 186 172 184 193
##  $ shoesize: int  39 44 41 43 45
##  $ age     : int  24 26 21 22 31
```

Accessing Matrices

- Dimension (#rows, #columns) or number of rows/columns

```
dim(d)

## [1] 5 4
```

```
nrow(d)

## [1] 5

ncol(d)

## [1] 4
```

- Access columns by name

```
d$height

## [1] 163 186 172 184 193

d[["height"]]

## [1] 163 186 172 184 193
```

- Accessing an individual element (notation: #row, #column)

```
d[1,2]

## [1] 163
```

Selecting Elements

- Using single condition to select a subset of rows

```
d[d$age > 25, ]
```

```
##    name height shoesize age
## 2 Robin    186       44  26
## 5 Jerry    193       45  31
```

```
d[d$age == 32, ]
```

```
## [1] name     height   shoesize age
## <0 rows> (or 0-length row.names)
```

- Connecting several conditions (`&` is and, `|` is or)

```
d[d$age < 25 & d$height <= 163, ]
```

```
##    name height shoesize age
## 1 Julia    163       39  24
```

Exercise: Selecting Elements

Question

- How to select all elements with `age` 26 or `shoesize` 45?
  - `d[d$age = 26 | d$shoesize = 45, ]`
  - `d[d$age == 26 | d$shoesize == 45, ]`
  - `d[d$age == 26 | d$shoesize == 45]`
  - `d[d$age == 26 & d$shoesize == 45, ]`

```
d[d$age == 26 | d$shoesize == 45, ]
```

```
##    name height shoesize age
## 2 Robin    186       44  26
## 5 Jerry    193       45  31
```

Exercise: Selecting Elements

Question

- How to select all elements with `age` 26 or `shoesize` 45?
  - `d[d$age = 26 | d$shoesize = 45, ]`
  - `d[d$age == 26 | d$shoesize == 45, ]`
  - `d[d$age == 26 | d$shoesize == 45]`
  - `d[d$age == 26 & d$shoesize == 45, ]`

```
d[d$age == 26 | d$shoesize == 45, ]

##    name height shoesize age
## 2 Robin    186       44  26
## 5 Jerry    193       45  31
```

Adding Columns and Column Names

- Adding columns

```
d[["heightInInch"]] <- d$height/2.51
d$heightInInch
```

```
## [1] 64.94024 74.10359 68.52590 73.30677 76.89243
```

- Getting column names via `colnames()`

```
colnames(d)
```

```
## [1] "name"         "height"       "shoesize"     "age"
## [5] "heightInInch"
```

- Updating column names

```
colnames(d) <- c("name", "waist", "weight", "shoes",
                 "books")
colnames(d)
```

```
## [1] "name"   "waist"  "weight" "shoes"  "books"
```

Outline

Managing Code Execution

- Control flow specifies order in which statements are executed
- Previous concepts can only execute R code in a linear fashion
- Control flow constructs can choose which execution path to follow

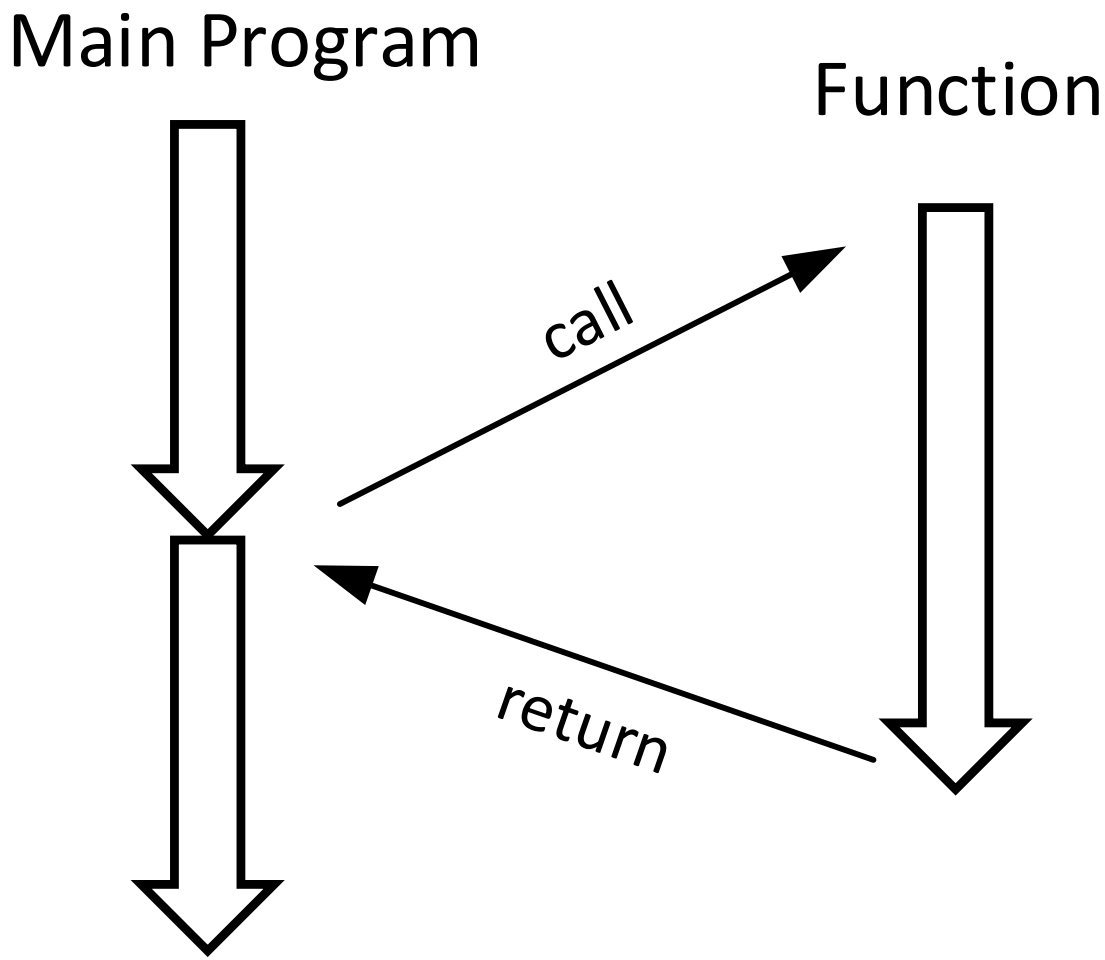
Functions: Combines sequence of statements into a self-contained task

Conditional expressions: Different computations according to a specific condition

Loops: Sequence of statements which may be executed more than once

Functions

- Functions avoid repeating the same code more than once
- Leave the current evaluation context to execute pre-defined commands



Functions

- Extend set of built-in functions with opportunity for customization
- Functions **can** consist of the following:
  1. Name to refer to (avoid existing function names in R)
  2. Function body is a sequence of statements
  3. Arguments define additional parameters passed to the function body
  4. Return value which can be used after executing the function
- Simple example

```
f <- function(x,y) {
  return(2*x + y^2)
}
f(-3, 5)

## [1] 19
```

Functions

- General syntax

```
functionname <- function(argument1, argument2, ...) {
  function_body
  return(value)
}
```

- Return value is the last evaluated expression
  → Alternative: set explicitly with `return(...)`
- Curly brackets can be omitted if the function contains only one statement (not recommended)
- Be cautious since the order of the arguments matters
- Values in functions are not printed in console
  → Remedy is `print(...)`

Examples of Functions

```
square <- function(x) x*x # last value is return value
square(10)

## [1] 100
```

```
cubic <- function(x) {
  # Print value to screen from inside the function
  print(c("Value: ", x, " Cubic: ", x*x*x))
  # no return value
}
cubic(10)

## [1] "Value: "  "10"         " Cubic: " "1000"
```

Examples of Functions

```
hello <- function() { # no arguments
  print("world")
}
hello()

## [1] "world"
```

```
my.mean <- function(x) {
  return (sum(x)/length(x))
}
my.mean(1:100)

## [1] 50.5
```

Scope in Functions

- Variables created inside a function only exists within it → local
- They are thus inaccessible from outside of the function
- Scope denotes when the name binding of variable is valid

```
x <- "A"
g <- function(x) {
  x <- "B"
  return(x)
}
x <- "C"
```

- What are the values?

```
g(x) # Return value of function x
x    # Value of x after function execution
```

- Solution

```
## [1] "B"
## [1] "C"
```

Scope in Functions

- Variables created inside a function only exists within it → local
- They are thus inaccessible from outside of the function
- Scope denotes when the name binding of variable is valid

```
x <- "A"
g <- function(x) {
  x <- "B"
  return(x)
}
x <- "C"
```

- What are the values?

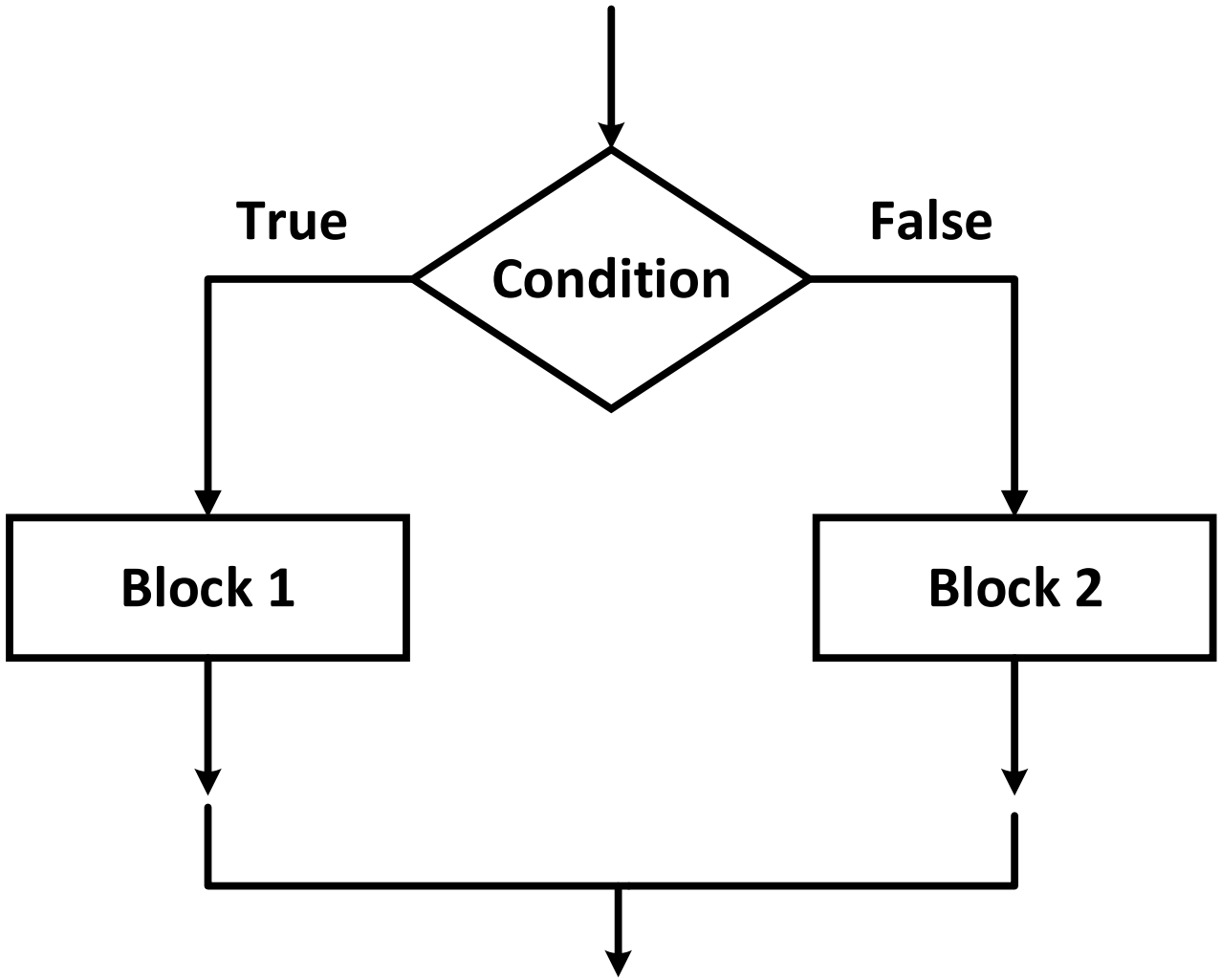
```
g(x) # Return value of function x
x    # Value of x after function execution
```

- Solution

```
## [1] "B"
## [1] "C"
```

If-Else Conditions

- Conditional execution requires a condition to be met



If-Else Conditions

- Keyword `if` with optional `else` clause
- General syntax:

if condition

```
if (condition) {
  statement1
}
```

If `condition` is true, then `statement` is executed

if-else condition

```
if (condition) {
  statement1
} else {
  statement2
}
```

If `condition` is true, then `statement1` is executed, otherwise `statement2`

If-Else Conditions

- Example

```
grade <- 2
if (grade <= 4) {
  print("Passed")
} else {
  print("Failed")
}

## [1] "Passed"
```

```
grade <- 5
if (grade <= 4) {
  print("Passed")
} else {
  print("Failed")
}

## [1] "Failed"
```

- Condition must be of length 1 and evaluate as either `TRUE` or `FALSE`

```
if (c(TRUE, FALSE)) { # don't do this!
  print("something")
}

## Warning in if (c(TRUE, FALSE)) {: the condition has
length > 1 and only the first element will be used

## [1] "something"
```

Else-If Clauses

- Multiple conditions can be checked with `else if` clauses
- The last `else` clause applies when no other conditions are fulfilled
- The same behavior can also be achieved with nested if-clauses

else-if clause

```
if (grade == 1) {
  print("very good")
} else if (grade == 2) {
  print("good")
} else {
  print("not a good grade")
}
```

Nested if-condition

```
if (grade == 1) {
  print("very good")
} else {
  if (grade == 2) {
    print("good")
  } else {
    print("not a good grade")
  }
}
```

If-Else Function

- As an alternative, one can also reach the same control flow via the function `ifelse(...)`

```
ifelse(condition, statement1, statement2)
# executes statement1 if condition is true,
# otherwise statement2
```

```
grade <- 2
ifelse(grade <= 4, "Passed", "Failed")

## [1] "Passed"
```

- `ifelse(...)` can also work with vectors as if it was applied to each element separately

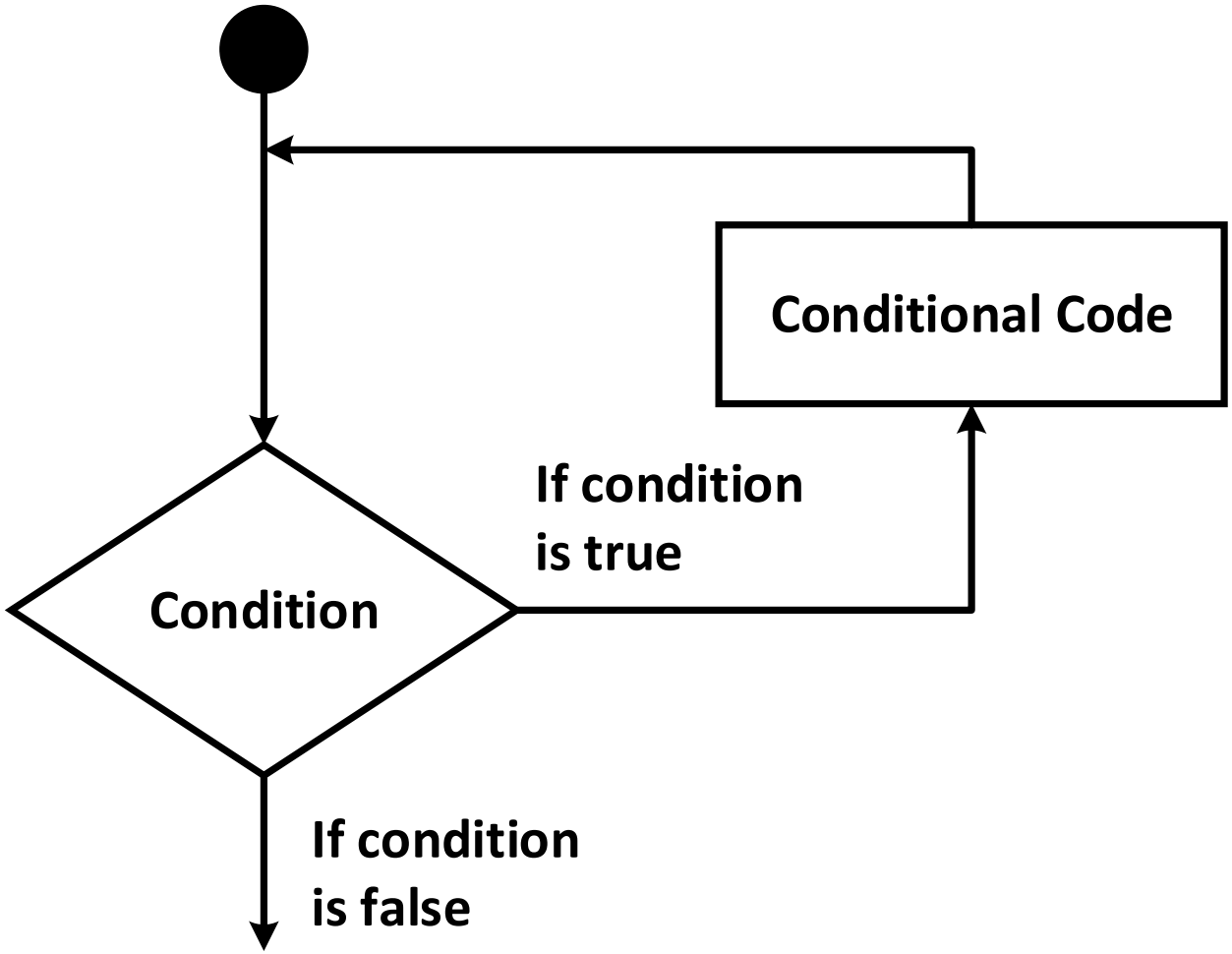
```
grades <- c(1, 2, 3, 4, 5)
ifelse(grades <= 4, "Passed", "Failed")

## [1] "Passed" "Passed" "Passed" "Passed" "Failed"
```

- This allows for the efficient comparison of vectors

For Loop

- `for` loops execute statements for a fixed number of repetitions



For Loop

- General syntax

```
for (counter in looping_vector){
  # code to be executed for each element in the sequence
}
```

- In every iteration of the loop, one value in the looping vector is assigned to the `counter` variable that can be used in the statements of the body of the loop.
- Examples

```
for (i in 4:7) {
  print(i)
}
```

```
## [1] 4
## [1] 5
## [1] 6
## [1] 7
```

```
a <- c()
for (i in 1:3){
  a[i] <- sqrt(i)
}
a
```

```
## [1] 1.000000 1.414214 1.732051
```

While Loop

- Loop where the number of iterations is controlled by a condition
- The condition is checked in every iteration
- When the condition is met, the loop body in curly brackets is executed
- General syntax

```
while (condition) {
  # code to be executed
}
```

- Examples

```
z <- 1
# same behavior as for loop
while (z <= 4) {
    print(z)
    z <- z + 1
}

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

```
z <- 1
# iterates all odd numbers
while (z <= 5) {
    z <- z + 2
    print(z)
}

## [1] 3
## [1] 5
## [1] 7
```

Outline

Extending R: Packages

- Most routines (from e. g. time series, statistical tests, plotting) are in so-called packages
- Packages must be downloaded & installed before usage
- When accessing routines, must be loaded via `library(package)`
- Installing packages by clicking:

In R Console

- Menu Packages
- Install package(s) . . .
- Choose arbitrary server
- Choose package

In R Studio

- Menu Tools
- Install packages
- Enter package name in middle input box
- Press Install

Exercise

Question

- You are doing an analysis in R and need to use the `summary()` function but you are not exactly sure how it works. Which of the following commands should you run?
  - `help(summary)`
  - `?summary`
  - `man(summary)`
  - `?summary()`

Any of the above commands work except for `man(summary)`. Make sure you always read the documentation so you know what functions do when you use them!

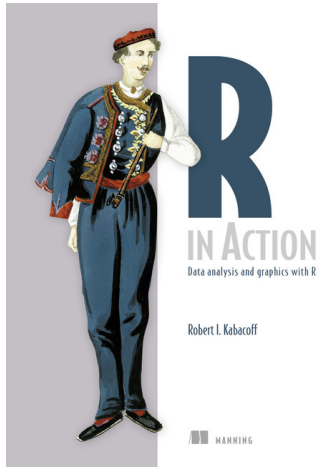
Outline

Tutorials on Using R

- Search Internet → many tutorials available online
- R Manual is the official introductory document
  → `http://cran.r-project.org/doc/manuals/R-intro.pdf`
- Helpful examples and demonstrations
  → `http://www.statmethods.net`
- Help pages in R describe parameters in detail, contain examples, but aim at advanced audience

Recommended Books

- R in Action: Data Analysis and Graphics with R
  (Manning, 2011, by Kabacoff)
  URL: `statmethods.net`
- Many more . . .



Summary: Commands

`+`, `-`, etc.	Algebraic operators
`&`, `|`, `<`, `<=`, etc.	Logic operators
`help(func)`	Help pages
`mean()`, `var()`	Functions on vectors
`sd()`	Standard deviation
`seq()`	Generate sequences
`d$column`	Accessing columns of a matrix
`read.csv()`	Reading text files