

# 1 - Intro to R

## Basics

Karsten Maurer, Eric Hare, and Susan VanderPlas

Iowa State University

# Overgrown Calculator

## Basic mathematical operators

*# Addition*

2 + 2

## [1] 4

*# Subtraction*

15 - 7

## [1] 8

*# Multiplication*

109\*23452

## [1] 2556268

*# Division*

3/7

## [1] 0.4286

# Overgrown Calculator

## Basic mathematical operators

*# Integer division*

7 **%%** 2

## [1] 3

*# Modulo operator (Remainder)*

7 **%%** 2

## [1] 1

*# Powers*

1.5<sup>**3**</sup>

## [1] 3.375

# Overgrown Calculator

## Other functions

- ▶ Exponentiation
  - ▶  $\exp(x)$
- ▶ Logarithms
  - ▶  $\log(x, \text{base} = 2)$
- ▶ Trigonometric functions
  - ▶  $\sin(x)$
  - ▶  $\text{asin}(x)$
  - ▶  $\cos(x)$
  - ▶  $\tan(x)$
- ▶ Variables
  - ▶ `MyAge <- 25`
  - ▶ `Height = "5ft4in"`

# Variables

## Variable Names

- ▶ Variable names can't start with a number
- ▶ R is case-sensitive
- ▶ Some common letters are used internally by R and should be avoided as variable names (c, q, t, C, D, F, T, I)
- ▶ There are reserved words that R won't let you use for variable names. (for, in, while, if, else, repeat, break, next)
- ▶ R will let you use the name of a predefined function. Try not to overwrite those though!
- ▶ ?make.names

# Basics

## Examples

```
data(tips, package="reshape2")  
(bill <- head(tips$total_bill))  
## [1] 16.99 10.34 21.01 23.68 24.59 25.29
```

```
log(bill)  
## [1] 2.833 2.336 3.045 3.165 3.202 3.230
```

```
bill[2]  
## [1] 10.34
```

```
sum(bill)  
## [1] 121.9
```

```
(bill_in_euros <- bill * .7982)  
## [1] 13.561 8.253 16.770 18.901 19.628 20.186
```

# Getting Help

- ▶ `help.start()`
- ▶ `help(command)`
- ▶ `?command`
- ▶ `help.search("command")`
- ▶ View the code! Type function name or use `getAnywhere("fun")`
- ▶ Google ("R + statistics + ...")
- ▶ StackOverflow

## Getting Out!

- ▶ `q()`

# R Reference Card

Download the reference card from `http://cran.r-project.org/doc/contrib/Short-refcard.pdf`  
Having this open or printed off and near you while working is helpful.



## Your Turn

- ▶ Find out how many rows and columns the 'iris' data set has. Figure out at least 2 ways to do this.  
Hint: "Variable Information" section on the first page of the reference card!
- ▶ Use rep to construct the following vector: 1 1 2 2 3 3 4 4 5 5  
Hint: "Data Creation" section of the reference card
- ▶ Use rep to construct this vector: 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5

# Vectors

- ▶ A vector is a list of values that are all the same type:

```
a <- 1:6  
a  
## [1] 1 2 3 4 5 6
```

- ▶ We can refer to an element of a vector by using its index: first, second, third:

```
b <- c(3, 4, 5, 1, 6, 2)  
b[3]  
## [1] 5  
  
b[6]  
## [1] 2
```

# Data Frames

## Quick Intro

- ▶ 'tips' is a data frame.
- ▶ Data frames hold data sets
- ▶ Not every column need be the same type - like an Excel spreadsheet
- ▶ Each column in a data frame is a vector - so each column needs to have values that are all the same type.
- ▶ We can access different columns using the \$ operator.

# Vectors

- ▶ As mentioned earlier, almost everything in R is a vector.

```
tip <- tips$tip  
bill <- tips$total_bill  
head(tip)  
## [1] 1.01 1.66 3.50 3.31 3.61 4.71
```

# Indexing

Sometimes we want to just grab part of a vector/matrix/dataframe. Vectors in R are 1-indexed - that is, we count 1, 2, 3.

```
head(tip)
## [1] 1.01 1.66 3.50 3.31 3.61 4.71
```

```
tip[1]
## [1] 1.01
```

```
c(1, 3, 5)
## [1] 1 3 5
```

```
tip[c(1, 3, 5)]
## [1] 1.01 3.50 3.61
```

```
tip[1:5]
## [1] 1.01 1.66 3.50 3.31 3.61
```

# Helpful functions

```
min(tip)
```

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

```
which.min(tip)
```

```
## [1] 68
```

```
tip[which.min(tip)]
```

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

```
size <- tips$size
```

```
which.min(size)
```

```
## [1] 68
```

```
which(size == min(size))
```

```
## [1] 68 83 112 223
```

# Vectors

Common operations and functions work on every element of a vector and return a new vector

```
head(tip * 100)
## [1] 101 166 350 331 361 471
```

```
head(bill * 0.18)
## [1] 3.058 1.861 3.782 4.262 4.426 4.552
```

```
head(round(bill * .18, 2))
## [1] 3.06 1.86 3.78 4.26 4.43 4.55
```

```
rate <- tip/bill
head(rate)
## [1] 0.05945 0.16054 0.16659 0.13978 0.14681 0.18624
```

# Logical Values

- ▶ R has built in support for logical values
- ▶ TRUE and FALSE are built in. T (for TRUE) and F (for FALSE) are supported but can be modified
- ▶ Logicals can result from a comparison using
  - ▶ `<`
  - ▶ `>`
  - ▶ `<=`
  - ▶ `>=`
  - ▶ `==`
  - ▶ `!=`



# Indexing

We can index using logical values as well

```
x <- c(2, 3, 5, 7)
x[c(T, F, F, T)]
## [1] 2 7
```

```
x > 3.5
## [1] FALSE FALSE  TRUE  TRUE
```

```
x[x > 3.5]
## [1] 5 7
```

```
x[c(T, F)]
## [1] 2 5
```

# Logical Values

## Examples

```
head(rate)
## [1] 0.05945 0.16054 0.16659 0.13978 0.14681 0.18624
```

```
sad_tip <- rate < 0.10
head(sad_tip)
## [1] TRUE FALSE FALSE FALSE FALSE FALSE
```

```
rate[sad_tip]
## [1] 0.05945 0.07180 0.07893 0.05680 0.09936 0.05643 0.09553
## [8] 0.07862 0.07296 0.08147 0.09984 0.09453 0.07718 0.07398
## [15] 0.06566 0.09560 0.09001 0.07746 0.08364 0.06653 0.08527
## [22] 0.08330 0.07937 0.03564 0.07358 0.08822 0.09820
```

# Indexing

We can modify subsets of vectors

```
x <- bill[1:5]  
x  
## [1] 16.99 10.34 21.01 23.68 24.59
```

```
x[1]  
## [1] 16.99
```

```
x[1] <- 20  
x  
## [1] 20.00 10.34 21.01 23.68 24.59
```

# Vectors

Elements of a vector all must be the same type.

```
head(rate)
## [1] 0.05945 0.16054 0.16659 0.13978 0.14681 0.18624
```

```
rate[sad_tip] <- ":-("
head(rate)
## [1] ":-(" "0.160541586073501"
## [3] "0.166587339362208" "0.139780405405405"
## [5] "0.146807645384303" "0.186239620403321"
```

All of the items in `rate` are now strings! That is sad!

# Data types

- ▶ Can use 'mode' or 'class' to find out information about variables
- ▶ 'str' is useful to find information about the structure of your data
- ▶ Many data types
  - ▶ integer
  - ▶ numeric
  - ▶ character
  - ▶ Date
  - ▶ factor

```
str(tips)
```

```
## 'data.frame': 244 obs. of 7 variables:
```

```
## $ total_bill: num 17 10.3 21 23.7 24.6 ...
```

```
## $ tip : num 1.01 1.66 3.5 3.31 3.61 4.71 2 3.12 1.96 3.23 ..
```

```
## $ sex : Factor w/ 2 levels "Female","Male": 1 2 2 2 1 2 2 2 2
```

```
## $ smoker : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 ..
```

```
## $ day : Factor w/ 4 levels "Fri","Sat","Sun",...: 3 3 3 3 3 3
```

```
## $ time : Factor w/ 2 levels "Dinner","Lunch": 1 1 1 1 1 1 1 1
```

```
## $ size : int 2 3 3 2 4 4 2 4 2 2 ...
```

# Data types

## Converting between types

Can convert between types using `as._____`

```
size <- head(tips$size)
```

```
size
```

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

```
as.character(size)
```

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

```
as.numeric("2")
```

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

```
as.factor(size)
```

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

```
## Levels: 2 3 4
```

# Data types

## Summary

```
size <- tips$size
summary(size)
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      1.00   2.00   2.00   2.57   3.00   6.00
```

```
summary(as.character(size))
##      Length      Class      Mode
##      244 character character
```

```
summary(as.factor(size))
##      1      2      3      4      5      6
##      4 156   38  37    5    4
```

# Functions

- ▶ Typical format: `foo(x, n = length(x), ...)`
- ▶ Some parameters have defaults set for you
- ▶ `...` is special. This passes along extra parameters to functions used inside the function
- ▶ Use `args(foo)` to view arguments for functions.



# Basic Statistical Functions

Using the basic functions we've learned it wouldn't be hard to compute some basic statistics.

```
(n <- length(tip))  
## [1] 244
```

```
(meantip <- sum(tip)/n)  
## [1] 2.998
```

```
(standdev <- sqrt( sum( (tip-meantip)^2 ) / (n-1) ) )  
## [1] 1.384
```

But we don't have to.

# Basic Statistical Functions

```
mean(tip)
## [1] 2.998
```

```
sd(tip)
## [1] 1.384
```

```
summary(tip)
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      1.00    2.00    2.90    3.00    3.56    10.00
```

```
quantile(tip, c(.025, .975))
##  2.5% 97.5%
## 1.176 6.462
```

# Distributions

- ▶ R has a lot of distributions built in.
- ▶ We can typically obtain:
  - ▶ Density value
  - ▶ CDF value
  - ▶ Inverse CDF value (percentiles)
  - ▶ Random deviate
- ▶ Normal (`norm`), Chi-square (`chisq`), F (`f`), T (`t`), Cauchy (`cauchy`), Poisson (`pois`), Binomial (`binom`), Negative Binomial (`nbinom`), Gamma (`gamma`), ..., lots more!
- ▶ `library(help = stats)`

# Distributions

## Examples

```
# Density - prepend with 'd'  
dnorm(0, mean = 0, sd = 1)  
## [1] 0.3989
```

```
# CDF - prepend with 'p'  
pnorm(2)  
## [1] 0.9772
```

```
# Inverse CDF - prepend with 'q'  
qnorm(.975)  
## [1] 1.96
```

```
# Random deviates - prepend with 'r'  
rnorm(5)  
## [1] 1.3975 1.8429 -0.7627 -0.2592 -0.2420
```

# Comparisons and Numeric Types

Be careful

```
2 == 2  
## [1] TRUE
```

```
sqrt(2)^2  
## [1] 2
```

```
sqrt(2)^2 == 2  
## [1] FALSE
```

```
options(digits = 22)  
sqrt(2)^2  
## [1] 2.0000000000000000444089
```

# Comparisons and Numeric Types

Be careful

```
sqrt(2)^2 == 2
```

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

```
all.equal(sqrt(2)^2, 2)
```

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

```
options(digits = 22)
```

```
2^64
```

```
## [1] 18446744073709551616
```

```
2^64 - 1
```

```
## [1] 18446744073709551616
```

```
# Uh oh! The 'gmp' or 'int64' packages provides better  
# integer precision if needed.
```

# Logical Operators

- ▶ & (elementwise AND)
- ▶ | (elementwise OR)

```
c(T, T, F, F) & c(T, F, T, F)
## [1] TRUE FALSE FALSE FALSE
```

```
c(T, T, F, F) | c(T, F, T, F)
## [1] TRUE TRUE TRUE FALSE
```

*# Which are big bills with a poor tip rate?*

```
id <- (bill > 40 & rate < .10)
```

```
tips[id,]
```

```
##      total_bill tip    sex smoker day   time size
## 103      44.30 2.5 Female   Yes  Sat  Dinner    3
## 183      45.35 3.5   Male   Yes  Sun  Dinner    3
## 185      40.55 3.0   Male   Yes  Sun  Dinner    2
```

?"&"

# Your Turn

Using the 'diamonds' dataset from ggplot2:

- ▶ Read up on the dataset (`?diamonds`)
- ▶ Create a variable for price/carat
- ▶ Compare the price of "Fair" diamonds and "Ideal" diamonds using summary. If you want to try to do a t-test to test for a difference in the mean of the prices.
- ▶ Plot price by carat (use `qplot` - go back to the motivating example for help with the syntax)
- ▶ Explore any interesting relationships you find.