R_Refresher

Lisa Chong

May 2018

Refresher on R

- Review notes
- Do practice problems to familiarize yourself with R

Working directory:

Where is working directory?

```
getwd()
```

Where should R look for directory? Set it

```
setwd("C:/Users/lisac/Documents/ISATEC/TropFishR_ISATEC")
```

Clear environment

```
rm(list = ls())
```

Assign

```
a <- 4 # '<-' means assign
b = 8
```

- not recommended to use '=' because it's used within functions

Help from R:

```
help("log")
help.search("logarithm")
# searching across packages
apropos("log")
# finds all functions of particular type
```

Assignment #1 – Use R to do the following:

1. $1+2(3+4)$
2. $ln(4^3 + 3^{2+1})$
3. $\sqrt{(4+3)(2+1)}$
4. $(\frac{1+2}{3+4})^2$

Answers

```
1+2*(3+4)
```

```
## [1] 15
```

```
log(4^3+3^(2+1))
```

```
## [1] 4.51086
```

```
sqrt((4+3)*(2+1))
```

```
## [1] 4.582576
```

```
((1+2)/(3+4))^2
```

```
## [1] 0.1836735
```

Data types

Every programming outcome in R can be stored as an **object**

- Numbers
- Characters (i.e. text or strings)
- Tables
- Vectors and matrices
- Plots
- Statistical output
- Functions

Need good names (i.e. don't use "data" as an object name". R will not like it. . .)

- valid variables must start with letter, but can be mix of letters, numbers, ".", and "_".

```r
# numeric
num <- 2.334
class(num)
```

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

Is 'num' a numeric class?

```r
is.double(num)
```

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

Coerce objects from one type to another

```
num <- as.integer(num)
```

What type of object is it?

```
typeof(num)
```

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

```r
# integer
int <- 5L

# logical
logical <- c("TRUE", "FALSE")
# either T or F
```

```r
# character
char <- "ja"

# complex
comp <- 3i
```

Vector:

```
(vec <- c(3,4,6,7,4,3,5))
```

```
## [1] 3 4 6 7 4 3 5
```

```
# parentheses around line prints it to the console
```

List:

```
lis <- list(x = 1:20,
            y = seq(from = 2, to = 12, by = 2))

# use ':' to create vector
```

The *seq* function allows more flexibility of creating consecutive numbers

- can also use : to create a vector in order

Vectors using *rep*

```
rep(3, times = 10)
```

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

```
# repeat 3 ten times

y <- 1:3
rep(y, length = 10)
```

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

```
# repeat y until 10 elements
```

Assignment #2

- Create vectors using `seq()`, `rep()`, and `c()` if necessary

1. Positive integers from 1 to 99
2. Odd integers between 1 and 99
3. Numbers 1,1,1,2,2,2,3,3,3
4. Fractions 1, 1/2, 1/3, 1/4 . . . 1/10

Answers

```
# 1)
1:99
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
## [24] 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
## [47] 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
## [70] 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86
## [93] 93 94 95 96 97 98 99
```

```
# 2)
seq(1,99, by = 2)
```

```
##  [1]  1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33
## [24] 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79
## [47] 93 95 97 99
```

Answers

```r
# 3)
rep(1:3, each = 3)
```

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

```r
# 4)
1/(1:10) # or
```

```
##  [1] 1.0000000 0.5000000 0.3333333 0.2500000 0.2000000 0
##  [8] 0.1250000 0.1111111 0.1000000
```

```r
paste("1/", 1:10, sep = "")
```

```
##  [1] "1/1"  "1/2"  "1/3"  "1/4"  "1/5"  "1/6"  "1/7"  "1
```

Matrix:

```
(mat <- matrix(data = 1:12, nrow = 4, ncol = 4))
```

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

Array:

```
(arr <- array(1:80, dim = c(4,10,2)))
```

```
## , , 1
##
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    1    5    9   13   17   21   25   29   33    37
## [2,]    2    6   10   14   18   22   26   30   34    38
## [3,]    3    7   11   15   19   23   27   31   35    39
## [4,]    4    8   12   16   20   24   28   32   36    40
##
## , , 2
##
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]   41   45   49   53   57   61   65   69   73    77
## [2,]   42   46   50   54   58   62   66   70   74    78
## [3,]   43   47   51   55   59   63   67   71   75    79
## [4,]   44   48   52   56   60   64   68   72   76    80
```

Data frame

```
df <- data.frame(x = 1:12, nrow = 4, ncol = 4)
head(df)
```

```
##   x nrow ncol
## 1 1    4    4
## 2 2    4    4
## 3 3    4    4
## 4 4    4    4
## 5 5    4    4
## 6 6    4    4
```

NULL

```
nu <- NULL
class(nu)
```

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

Missing values

```
(na_vec <- c(4, -Inf, 98, 23, NA, NaN, Inf))
```

```
## [1]    4 -Inf   98   23   NA  NaN  Inf
```

NA vs NaN

- NA: not available
- NaN: not a number (always numeric)

Easy functions

```
mean(vec)
```

```
## [1] 4.571429
```

```
median(vec)
```

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

```
quantile(vec)
```

```
##   0%  25%  50%  75% 100%
##  3.0  3.5  4.0  5.5  7.0
```

```
summary(vec)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.000   3.500   4.000   4.571   5.500   7.000
```

Easy functions

```
min(vec)
```

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

```
max(vec)
```

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

```
var(vec)
```

```
## [1] 2.285714
```

```
sd(vec)
```

```
## [1] 1.511858
```

Subsetting vectors

```
x <- c(3,4,2,1,10,7)
x[1] # include index 1
```

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

```
x[3]
```

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

```
x[1:5] # use vector of indices to select multiple
```

```
## [1]  3  4  2  1 10
```

```
x[-c(2,4)] # negative means exclude items
```

```
## [1]  3  2 10  7
```

Logical operators

- <
- >
- >= “greater than or equal to”
- <= “less than or equal to”
- != “not equal”
- == “equals” (must be two == !)
- & “and”
- | “or”
- ! “not”

Logical operators

```
cat <- c("Meow", "Bark", "Bark",
         "Meow", "Meow", "Bark", "Meow")

which(cat == "Meow") # which elements are "Meow"
```

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

```
any(cat == "Meow") # Are any elements "Meow"
```

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

```
all(cat == "Meow") # Are all elements "Meow"
```

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

`length()` function

- returns number of elements in vector

```
length(vec)
```

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

Assignment #3

y <- c(3,2,15,-1,22,1,9,17,5)

1. Display first and last values
2. Last value for vector of any length
3. Values greater than mean of y?
4. Any value equal to mean?
5. Display positions (indices) of values greater than median

Answers

```r
y <- c(3,2,15,-1,22,1,9,17,5)

# 1)
y[c(1,9)]
```

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

```r
# 2)
y[length(y)]
```

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

```r
# 3)
y > mean(y)
```

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

Answers

```r
# 4)
y == mean(y)
```

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

```r
# 5)
y[y > median(y)]
```

```
## [1] 15 22  9 17
```

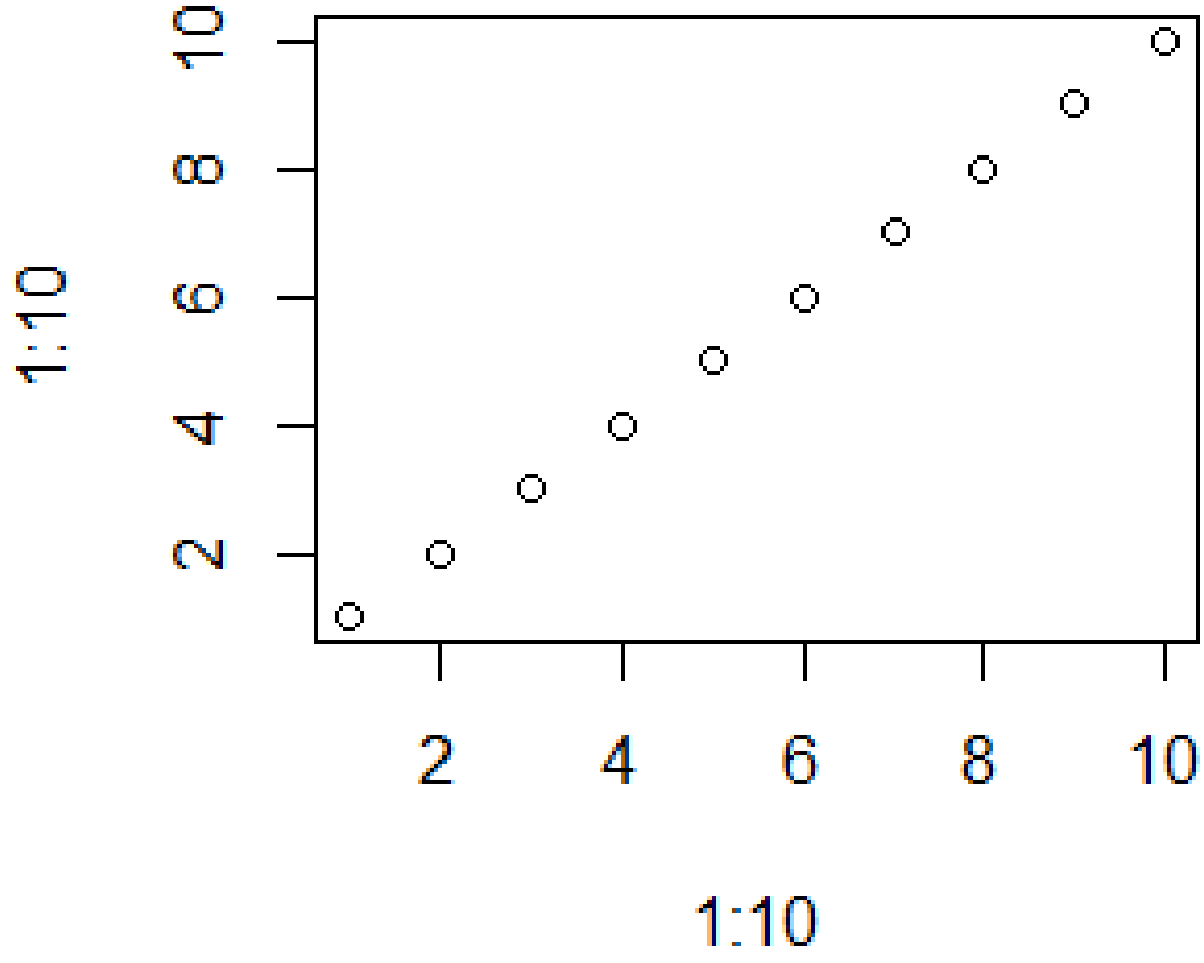
Reading in data

- `read.csv()`: reads Excel worksheets (in csv format) or other comma-separated data
- `read.csv2()`: changes separater (changes how decimal point is coded)
- `read.table()`: reads into data frame

Reading in data - read.table() arguments

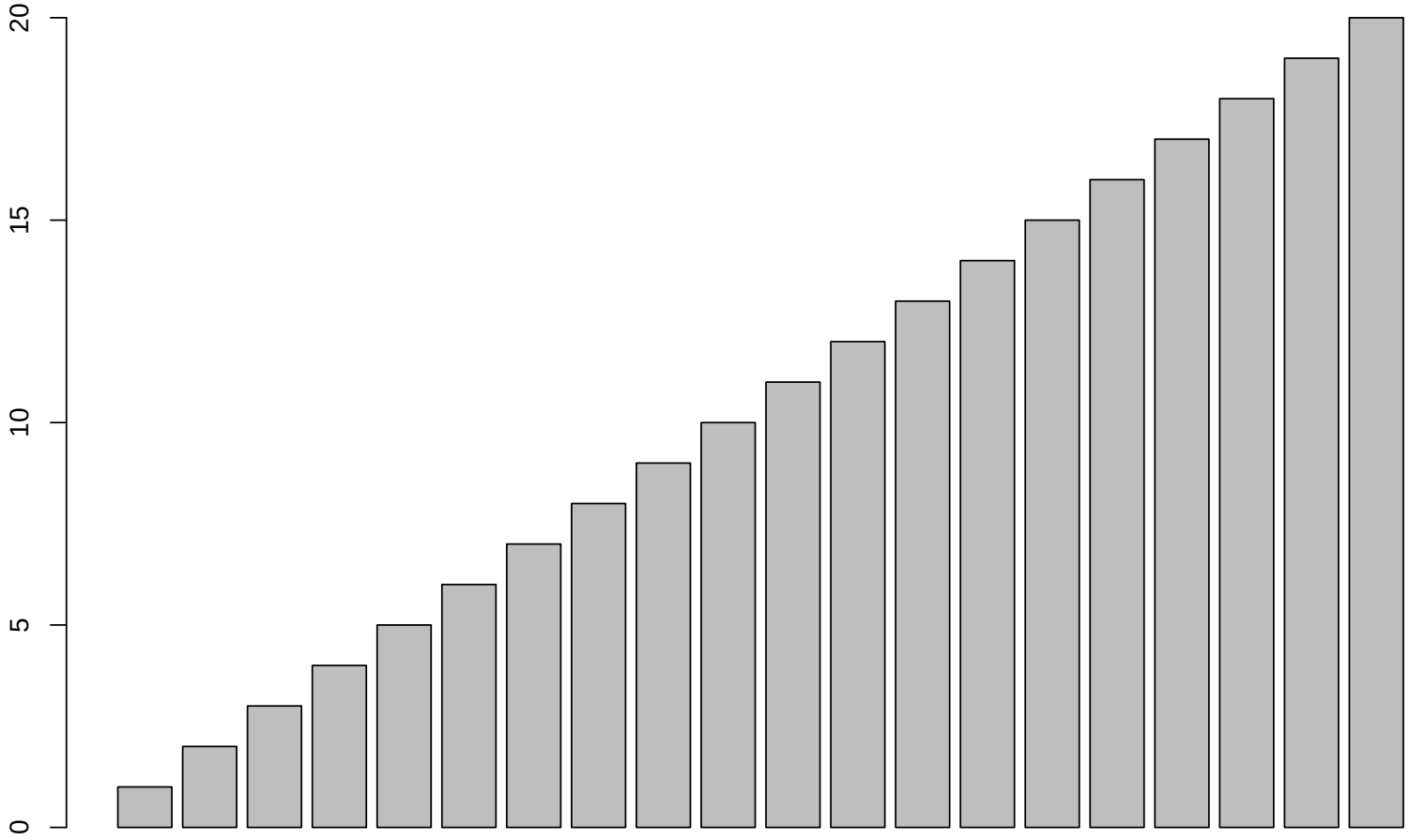
- header = T ; first row names for columns
- sep = "" ; how are entries separated
- na.strings = NA ; which values treated as NAs
- skip = 0 ; number of lines to skip before reading data
- nrows = 1 ; number of lines to read (-1 means all)
- col.names = c("a", "b") ; names for columns

Plotting in R



Plotting in R

```
barplot(1:20)
```



Plotting in R

```
hist(rnorm(3000), breaks = 50)
```