https://intro2r.com/ Chapter 2

CSCI 297b, Spring 2023

April 25, 2023

R basics

- R is case sensitive. `anova` is not the same as `Anova`
- Anything following # is a comment and is ignored by R
- Comments should be used liberally
- Commands are separated by a newline or a semicolon ;
- A continuation prompt, +, means the previous line is not finished
- If execution hangs and does not stop, try the escape key or the stop button

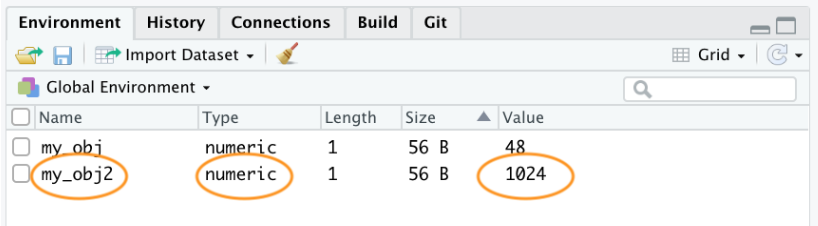
Some builtin R functions

```
log(1)              # logarithm to base e
## [1] 0
log10(1)            # logarithm to base 10
## [1] 0
exp(1)              # natural antilog
## [1] 2.718282
sqrt(4)             # square root
## [1] 2
4^2                   # 4 to the power of 2
## [1] 16
pi                    # not a function but useful
## [1] 3.141593
```

Objects and assignment

```
> my_obj <- 1729
> my_obj2 <- "R is cool"
> my_obj
[1] 1729
> my_obj3 <- my_obj / 2
> my_obj3
[1] 864.5
> my_obj4 <- my_obj + my_obj3
> my_obj4
[1] 2593.5
> my_obj5 <- my_obj + my_obj2
Error in my_obj + my_obj2 : non-numeric argument to binary opera
>
```

The Environment Tab



□	Name	Type	Length	Size	Value
□	my_obj	numeric	1	56 B	48
□	my_obj2	numeric	1	56 B	1024

Naming Objects

> *There are two hard problems in computer science: cache invalidation, naming things, and off-by-1 errors.*
>
> — *Leon Bambrick*

Naming Objects

> *There are two hard problems in computer science: cache invalidation, naming things, and off-by-1 errors.*
>
> — *Leon Bambrick*

Two often conflicting goals:

- Short
- Meaningful

Name conventions

```
output_summary <- "my analysis"     # snake case
output.summary <- "my analysis"     # dot case
outputSummary <- "my analysis"      # camel case
OutputSummary <- "my analysis"      # Pascal case
output-summary <- "my analysis"     # kebab case
```

- Snake case used by textbook
- Google style recommends Pascal for function names
- Kebab case is illegal in R
- Dots illegal in many other languages
- Camel case is my favorite

Don't use existing names

```
data <- read.table("mydatafile", header = TRUE)
#data is a function!
```

Do exercise 2, part 1

The `c()` function

```
my_vec <- c(2,3,1,6,4,3,3,7)
mean(my_vec)    # returns the mean of my_vec
## [1] 3.625
var(my_vec)     # returns the variance of my_vec
## [1] 3.982143
sd(my_vec)      # returns the standard deviation of my_vec
## [1] 1.995531
length(my_vec)  # returns the number of elements in my_vec
## [1] 8
```

Sequences

```
my_seq <- 1:10     # create regular sequence
my_seq
##  [1]  1  2  3  4  5  6  7  8  9 10
my_seq2 <- 10:1    # in decending order
my_seq2
##  [1] 10  9  8  7  6  5  4  3  2  1
my_seq2 <- seq(from = 1, to = 5, by = 0.5)
my_seq2
## [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
my_seq3 <- rep(2, times = 10)   # repeats 2, 10 times
my_seq3
##  [1] 2 2 2 2 2 2 2 2 2 2
my_seq4 <- rep("abc", times = 3)    # repeats ’abc’ 3 times
my_seq4
## [1] "abc" "abc" "abc"
```

Sequences

```
my_seq5 <- rep(1:5, times = 3)  # repeats the series 1 to
                                # 5, 3 times
my_seq5
##  [1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
my_seq6 <- rep(1:5, each = 3)   # repeats each element of
                                # the series 3 times
my_seq6
##  [1] 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5
my_seq7 <- rep(c(3, 1, 10, 7), each = 3) # repeats each
                                         # element of the
                                         # series 3 times
my_seq7
##  [1]  3  3  3  1  1  1 10 10 10  7  7  7

## Alternative approach:
in_vec <- c(3, 1, 10, 7)
my_seq7 <- rep(in_vec, each = 3)
my_seq7
##  [1]  3  3  3  1  1  1 10 10 10  7  7  7
```

Positional indexing

```
my_vec        # remind ourselves what my_vec looks like
## [1] 2 3 1 6 4 3 3 7
my_vec[3]     # extract the 3rd value
## [1] 1

# if you want to store this value in another object
val_3 <- my_vec[3]
val_3
## [1] 1
my_vec[c(1, 5, 6, 8)]
## [1] 2 4 3 7
my_vec[3:8]
## [1] 1 6 4 3 3 7
```

Logical indexing

```
my_vec        # remind ourselves what my_vec looks like
## [1] 2 3 1 6 4 3 3 7
my_vec[my_vec > 4]
## [1] 6 7
my_vec > 4
## [1] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE
my_vec[c(FALSE, FALSE, FALSE, TRUE, FALSE, FALSE, FALSE, TRUE)]
## [1] 6 7
```

Logical indexing

```
my_vec        # remind ourselves what my_vec looks like
## [1] 2 3 1 6 4 3 3 7
my_vec[my_vec >= 4]        # values greater or equal to 4
## [1] 6 4 7
my_vec[my_vec < 4]         # values less than 4
## [1] 2 3 1 3 3
my_vec[my_vec <= 4]        # values less than or equal to 4
## [1] 2 3 1 4 3 3
my_vec[my_vec == 4]        # values equal to 4
## [1] 4
my_vec[my_vec != 4]        # values not equal to 4
## [1] 2 3 1 6 3 3 7
```

Boolean expressions

```
my_vec        # remind ourselves what my_vec looks like
## [1] 2 3 1 6 4 3 3 7
val26 <- my_vec[my_vec < 6 & my_vec > 2]
val26
## [1] 3 4 3 3
val63 <- my_vec[my_vec > 6 | my_vec < 3]
val63
## [1] 2 1 7
```

Replacing elements

```
my_vec        # remind ourselves what my_vec looks like
## [1] 2 3 1 6 4 3 3 7

## replace the 4th element with 500
my_vec[4] <- 500
my_vec
## [1]   2   3   1 500   4   3   3   7

# replace the 6th and 7th element with 100
my_vec[c(6, 7)] <- 100
my_vec
## [1]   2   3   1 500   4 100 100   7

# replace element that are less than or equal to 4 with 1000
my_vec[my_vec <= 4] <- 1000
my_vec
## [1] 1000 1000 1000  500 1000  100  100    7
```

Do exercise 2, part 2

Sorting elements

```
my_vec
## [1] 1000 1000 1000  500 1000  100  100    7

vec_sort <- sort(my_vec)
vec_sort
## [1]    7  100  100  500 1000 1000 1000 1000

vec_sort2 <- sort(my_vec, decreasing = TRUE)
vec_sort2
## [1] 1000 1000 1000 1000  500  100  100    7

vec_sort3 <- rev(sort(my_vec))
vec_sort3
## [1] 1000 1000 1000 1000  500  100  100    7
```

Ordering elements

```
height <- c(180, 155, 160, 167, 181)
height
## [1] 180 155 160 167 181

p.names <- c("Joanna", "Charlotte", "Helen", "Karen", "Amy")
p.names
## [1] "Joanna"    "Charlotte" "Helen"     "Karen"     "Amy"

height_ord <- order(height)
height_ord
## [1] 2 3 4 1 5

 height[height_ord]
## [1] 155 160 167 180 181

names_ord <- p.names[height_ord]
names_ord
## [1] "Charlotte" "Helen"     "Karen"     "Joanna"    "Amy"
```

Vectorization

```
# create a vector
my_vec2 <- c(3, 5, 7, 1, 9, 20)

# multiply each element by 5
my_vec2 * 5
## [1]  15  25  35   5  45 100

# create a second vector
my_vec3 <- c(17, 15, 13, 19, 11, 0)

# add both vectors
my_vec2 + my_vec3
## [1] 20 20 20 20 20 20

# multiply both vectors
my_vec2 * my_vec3
## [1] 51 75 91 19 99  0
```

Vectorization recycling

```
my_vec2 <- c(3, 5, 7, 1, 9, 20)
my_vec4 <- c(1, 2)

# add both vectors - quiet recycling!
my_vec2 + my_vec4
## [1]  4  7  8  3 10 22
```

Missing data

```
temp  <- c(7.2, NA, 7.1, 6.9, 6.5, 5.8, 5.8, 5.5, NA, 5.5)
temp
##  [1] 7.2  NA 7.1 6.9 6.5 5.8 5.8 5.5  NA 5.5

mean_temp <- mean(temp)
mean_temp
## [1] NA

mean_temp <- mean(temp, na.rm = TRUE)
mean_temp
## [1] 6.2875
```

- Some functions may deal with NAs differently.
- Always consult the documentation.

R help

```
## show the help page:
help("mean")
?mean

## search all help pages:
help.search("mean")
??mean
```

- Also use the Help tab in RStudio
- Usually the most helpful part of a help page is the examples.

R help

```
apropos("mean")
##  [1] ".colMeans"     ".rowMeans"     "colMeans"      "kmeans"
##  [5] "mean"          "mean_temp"     "mean.Date"     "mean.de
##  [9] "mean.difftime" "mean.POSIXct"  "mean.POSIXlt"  "rowMean
## [13] "vec_mean"      "weighted.mean"
```

- Find all functions with ”mean” in their name.

```
RSiteSearch("regression")
```

- Search for keywords and phrases in function help pages and vignettes for all CRAN packages, and in CRAN task views.

General R resources

- `https://cran.r-project.org/other-docs.html`
  R-Project: User contributed documentation
- `https://journal.r-project.org/`
  The R Journal: Journal of the R project for statistical computing
- `http://swirlstats.com/`
  Swirl: An R package that teaches you R from within R
- `https://www.rstudio.com/resources/cheatsheets/`
  RStudio's printable cheatsheets
- `http://rseek.org/`
  Rseek A custom Google search for R-related sites

Getting help

- `https://www.google.com/`

  Google it!: Try Googling any error messages you get. It's not cheating and everyone does it! You'll be surprised how many other people have probably had the same problem and solved it.

- `http://stackoverflow.com/questions/tagged/r`

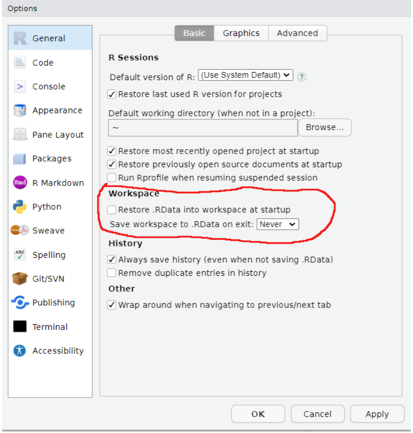
  Stack Overflow: There are many thousands of questions relevant to R on Stack Overflow. Here are the most popular ones, ranked by vote. Make sure you search for similar questions before asking your own, and make sure you include a reproducible example to get the most useful advice. A reproducible example is a minimal example that lets others who are trying to help you to see the error themselves.

Saving stuff

- Normally the only thing you need to save is your R script.
- Put everything you do in the script.
- Save it, and the next time you start just source it.

Global options

- Tools → Global Options ...
- Turn off `.RData` save and restore
- This prevents new R sessions from being influenced by previous R sessions.



R Sessions

Default version of R: (Use System Default)

☑ Restore last used R version for projects

Default working directory (when not in a project): ~ Browse...

☑ Restore most recently opened project at startup

☑ Restore previously open source documents at startup

□ Run Rprofile when resuming suspended session

Workspace

□ Restore .RData into workspace at startup

Save workspace to .RData on exit: Never

History

☑ Always save history (even when not saving .RData)

□ Remove duplicate entries in history

Other

☑ Wrap around when navigating to previous/next tab

Saving objects

- Occasionally you want to save an object rather than recompute it.
- Some objects may take minutes or hours to compute.

```
## save a single object to a file
save(nameOfObject, file = "name_of_file.RData")

## save all the objects in your workspace in a single file
save.image(file = "name_of_file.RData")

## reload whatever you saved
load(file = "name_of_file.RData")
```

Do exercise 2, part 3