

<https://intro2r.com/> Chapter 2

CSCI 297b, Spring 2023

April 21, 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

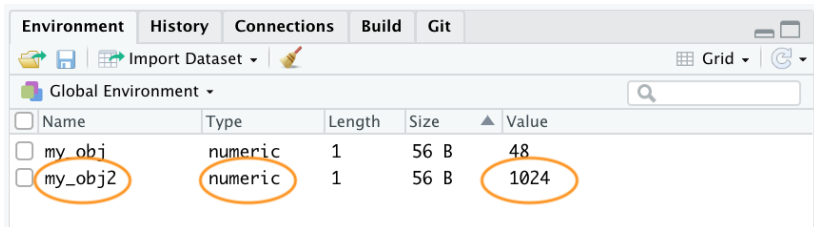
## Some builtin R functions

```
1 log(1)                # logarithm to base e
2 ## [1] 0
3 log10(1)              # logarithm to base 10
4 ## [1] 0
5 exp(1)               # natural antilog
6 ## [1] 2.718282
7 sqrt(4)              # square root
8 ## [1] 2
9 4^2                  # 4 to the power of 2
10 ## [1] 16
11 pi                  # not a function but useful
12 ## [1] 3.141593
```

## Objects and assignment

```
1 > my_obj <- 1729
2 > my_obj2 <- "R is cool"
3 > my_obj
4 [1] 1729
5 > my_obj3 <- my_obj / 2
6 > my_obj3
7 [1] 864.5
8 > my_obj4 <- my_obj + my_obj3
9 > my_obj4
10 [1] 2593.5
11 > my_obj5 <- my_obj + my_obj2
12 Error in my_obj + my_obj2 : non-numeric
    argument to binary operator
13 >
```

# The Environment Tab



The screenshot shows the RStudio Environment Tab with the following table of objects:

<input type="checkbox"/>	Name	Type	Length	Size	Value
<input type="checkbox"/>	my_obj	numeric	1	56 B	48
<input type="checkbox"/>	my_obj2	numeric	1	56 B	1024

The values 1024 and the word 'numeric' are circled in orange in the original image.

# 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

```
1 output_summary <- "my analysis"      # snake case
2 output.summary <- "my analysis"      # dot case
3 outputSummary <- "my analysis"       # camel case
4 OutputSummary <- "my analysis"       # Pascal case
5 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

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

Do exercise 2, part 1

## The `c()` function

```
1 my_vec <- c(2,3,1,6,4,3,3,7)
2 mean(my_vec)      # returns the mean of my_vec
3 ## [1] 3.625
4 var(my_vec)       # returns the variance of my_vec
5 ## [1] 3.982143
6 sd(my_vec)        # returns the standard deviation of my_vec
7 ## [1] 1.995531
8 length(my_vec)    # returns the number of elements in my_vec
9 ## [1] 8
```

# Sequences

```
1 my_seq <- 1:10      # create regular sequence
2 my_seq
3 ## [1]  1  2  3  4  5  6  7  8  9 10
4 my_seq2 <- 10:1     # in decending order
5 my_seq2
6 ## [1] 10  9  8  7  6  5  4  3  2  1
7 my_seq2 <- seq(from = 1, to = 5, by = 0.5)
8 my_seq2
9 ## [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
10 my_seq3 <- rep(2, times = 10) # repeats 2, 10 times
11 my_seq3
12 ## [1] 2 2 2 2 2 2 2 2 2 2
13 my_seq4 <- rep("abc", times = 3) # repeats 'abc' 3 times
14 my_seq4
15 ## [1] "abc" "abc" "abc"
```

# Sequences

```
1 my_seq5 <- rep(1:5, times = 3) # repeats the series 1 to
2                               # 5, 3 times
3 my_seq5
4 ## [1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
5 my_seq6 <- rep(1:5, each = 3)  # repeats each element of
6                               # the series 3 times
7 my_seq6
8 ## [1] 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5
9 my_seq7 <- rep(c(3, 1, 10, 7), each = 3) # repeats each
10                                           # element of the
11                                           # series 3 times
12 my_seq7
13 ## [1] 3 3 3 1 1 1 10 10 10 7 7 7
14
15 ## Alternative approach:
16 in_vec <- c(3, 1, 10, 7)
17 my_seq7 <- rep(in_vec, each = 3)
18 my_seq7
19 ## [1] 3 3 3 1 1 1 10 10 10 7 7 7
```

# Positional indexing

```
1 my_vec          # remind ourselves what my_vec looks like
2 ## [1] 2 3 1 6 4 3 3 7
3 my_vec[3]       # extract the 3rd value
4 ## [1] 1
5
6 # if you want to store this value in another object
7 val_3 <- my_vec[3]
8 val_3
9 ## [1] 1
10 my_vec[c(1, 5, 6, 8)]
11 ## [1] 2 4 3 7
12 my_vec[3:8]
13 ## [1] 1 6 4 3 3 7
```

# Logical indexing

```
1 my_vec          # remind ourselves what my_vec looks like
2 ## [1] 2 3 1 6 4 3 3 7
3 my_vec[my_vec > 4]
4 ## [1] 6 7
5 my_vec > 4
6 ## [1] FALSE FALSE FALSE TRUE FALSE FALSE FALSE TRUE
7 my_vec[c(FALSE, FALSE, FALSE, TRUE, FALSE, FALSE, FALSE,
8           TRUE)]
8 ## [1] 6 7
```

# Logical indexing

```
1 my_vec          # remind ourselves what my_vec looks like
2 ## [1] 2 3 1 6 4 3 3 7
3 my_vec[my_vec >= 4]      # values greater or equal to 4
4 ## [1] 6 4 7
5 my_vec[my_vec < 4]       # values less than 4
6 ## [1] 2 3 1 3 3
7 my_vec[my_vec <= 4]      # values less than or equal to 4
8 ## [1] 2 3 1 4 3 3
9 my_vec[my_vec == 4]      # values equal to 4
10 ## [1] 4
11 my_vec[my_vec != 4]     # values not equal to 4
12 ## [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

```
1 my_vec          # remind ourselves what my_vec looks like
2 ## [1] 2 3 1 6 4 3 3 7
3
4 ## replace the 4th element with 500
5 my_vec[4] <- 500
6 my_vec
7 ## [1] 2 3 1 500 4 3 3 7
8
9 # replace the 6th and 7th element with 100
10 my_vec[c(6, 7)] <- 100
11 my_vec
12 ## [1] 2 3 1 500 4 100 100 7
13
14 # replace element that are less than or equal to 4 with 1000
15 my_vec[my_vec <= 4] <- 1000
16 my_vec
17 ## [1] 1000 1000 1000 500 1000 100 100 7
```

## Do exercise 2, part 2

## Sorting elements

```
1 my_vec
2 ## [1] 1000 1000 1000 500 1000 100 100 7
3
4 vec_sort <- sort(my_vec)
5 vec_sort
6 ## [1] 7 100 100 500 1000 1000 1000 1000
7
8 vec_sort2 <- sort(my_vec, decreasing = TRUE)
9 vec_sort2
10 ## [1] 1000 1000 1000 1000 500 100 100 7
11
12 vec_sort3 <- rev(sort(my_vec))
13 vec_sort3
14 ## [1] 1000 1000 1000 1000 500 100 100 7
```

## Ordering elements

```
1 height <- c(180, 155, 160, 167, 181)
2 height
3 ## [1] 180 155 160 167 181
4
5 p.names <- c("Joanna", "Charlotte", "Helen", "Karen", "Amy")
6 p.names
7 ## [1] "Joanna"      "Charlotte" "Helen"      "Karen"      "Amy"
8
9 height_ord <- order(height)
10 height_ord
11 ## [1] 2 3 4 1 5
12
13 height[height_ord]
14 ## [1] 155 160 167 180 181
15
16 names_ord <- p.names[height_ord]
17 names_ord
18 ## [1] "Charlotte" "Helen"      "Karen"      "Joanna"      "Amy"
```

# Vectorization

```
1 # create a vector
2 my_vec2 <- c(3, 5, 7, 1, 9, 20)
3
4 # multiply each element by 5
5 my_vec2 * 5
6 ## [1] 15 25 35 5 45 100
7
8 # create a second vector
9 my_vec3 <- c(17, 15, 13, 19, 11, 0)
10
11 # add both vectors
12 my_vec2 + my_vec3
13 ## [1] 20 20 20 20 20 20
14
15 # multiply both vectors
16 my_vec2 * my_vec3
17 ## [1] 51 75 91 19 99 0
```

# Vectorization recycling

```
1 my_vec2 <- c(3, 5, 7, 1, 9, 20)
2 my_vec4 <- c(1, 2)
3
4 # add both vectors – quiet recycling!
5 my_vec2 + my_vec4
6 ## [1]  4  7  8  3 10 22
```

## Missing data

```
1 temp <- c(7.2, NA, 7.1, 6.9, 6.5, 5.8, 5.8, 5.5, NA, 5.5)
2 temp
3 ## [1] 7.2 NA 7.1 6.9 6.5 5.8 5.8 5.5 NA 5.5
4
5 mean_temp <- mean(temp)
6 mean_temp
7 ## [1] NA
8
9 mean_temp <- mean(temp, na.rm = TRUE)
10 mean_temp
11 ## [1] 6.2875
```

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



# R help

```
1 ## show the help page:  
2 help("mean")  
3 ?mean  
4  
5 ## search all help pages:  
6 help.search("mean")  
7 ??mean
```

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

# R help

```
1 apropos("mean")
2 ## [1] ".colMeans"      ".rowMeans"      "colMeans"      "kmeans"
3 ## [5] "mean"           "mean.temp"      "mean.Date"      "mean.default"
4 ## [9] "mean.difftime"  "mean.POSIXct"   "mean.POSIXlt"   "rowMeans"
5 ## [13] "vec.mean"       "weighted.mean"
```

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

```
1 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

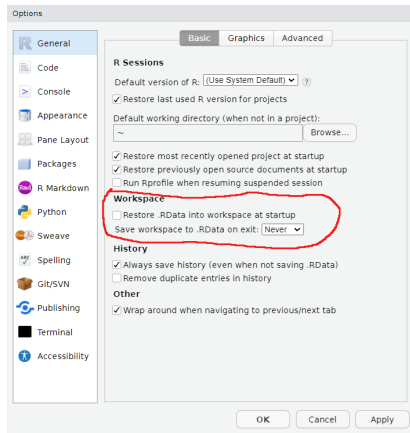
- <https://www.google.com/>  
Google it!: Try Googling any error messages you get. It's not cheating and everyone does it! You'll be surprised how many other people have probably had the same problem and solved it.
- <http://stackoverflow.com/questions/tagged/r>  
Stack Overflow: There are many thousands 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.



# Saving objects

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

```
1 ## save a single object to a file
2 save(nameOfObject, file = "name_of_file.RData")
3
4 ## save all the objects in your workspace in a single file
5 save.image(file = "name_of_file.RData")
6
7 ## reload whatever you saved
8 load(file = "name_of_file.RData")
```

Do exercise 2, part 3