Coding Basics and Importing Data

Patrick Mathias

Lesson 2

DLMP Fall 2021

Lesson Goals

1. Learn some fundamental of coding

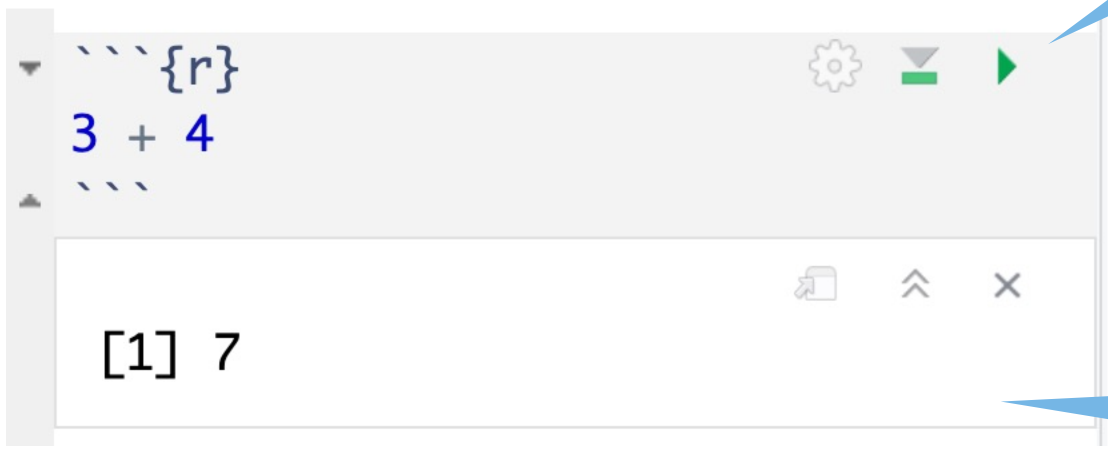
Lesson Objectives

1. Define and use functions
2. Define and create objects in the environment
3. Install and load packages
4. Import data into R
5. Interact with a dataframe

The Basics of Coding

The Basics of Coding: Calculation

- R is a calculator!



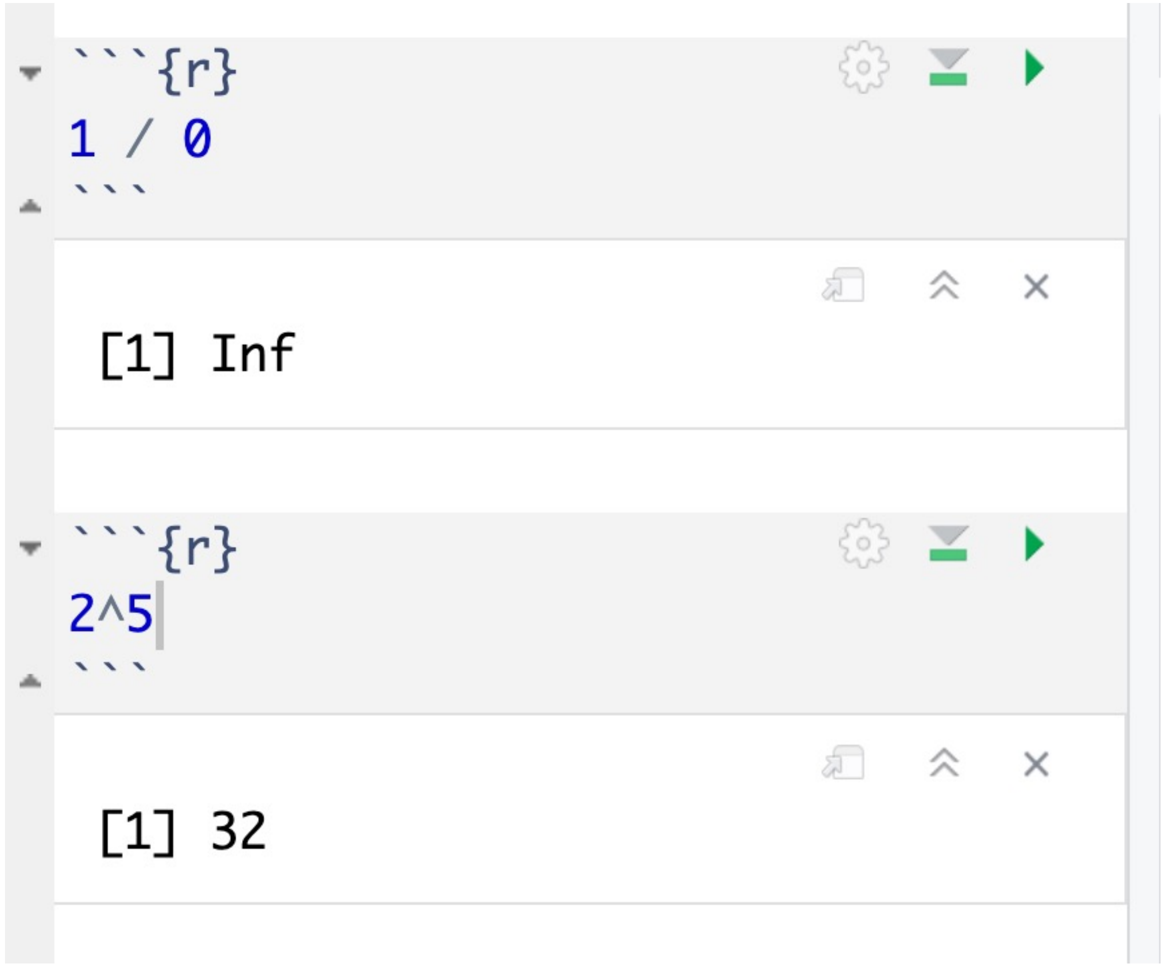
~~~
```{r}
3 + 4
```
~~~

```
[1] 7
```

press play button to execute code

answer returned below

Common arithmetic operations



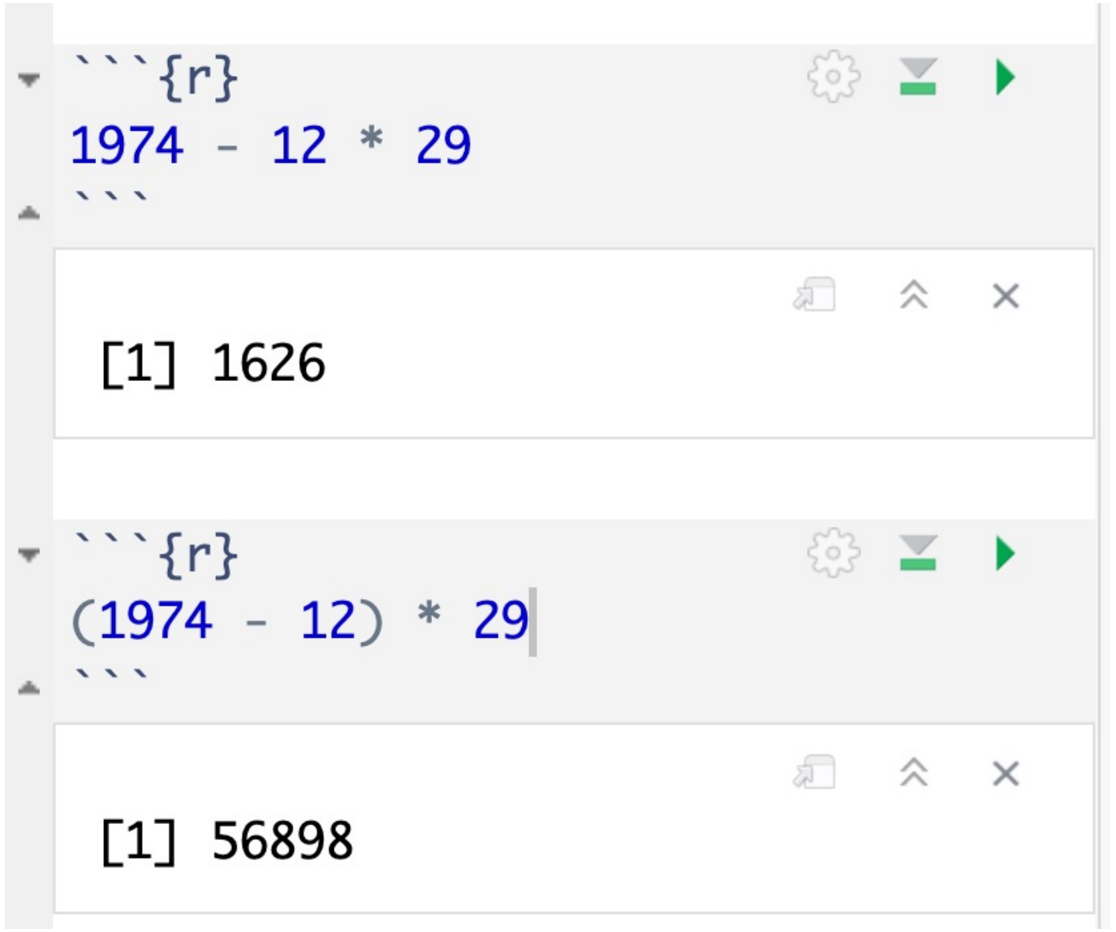
~~~
```{r}
1 / 0
```
~~~

```
[1] Inf
```

~~~
```{r}
2^5
```
~~~

```
[1] 32
```

Don’t forget those math rules



```{r}
1974 - 12 * 29
```

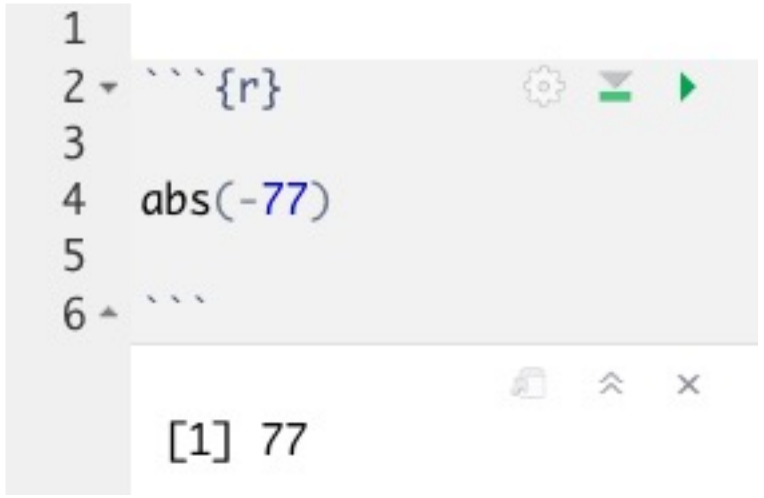
```
[1] 1626
```

```{r}
(1974 - 12) * 29
```

```
[1] 56898
```

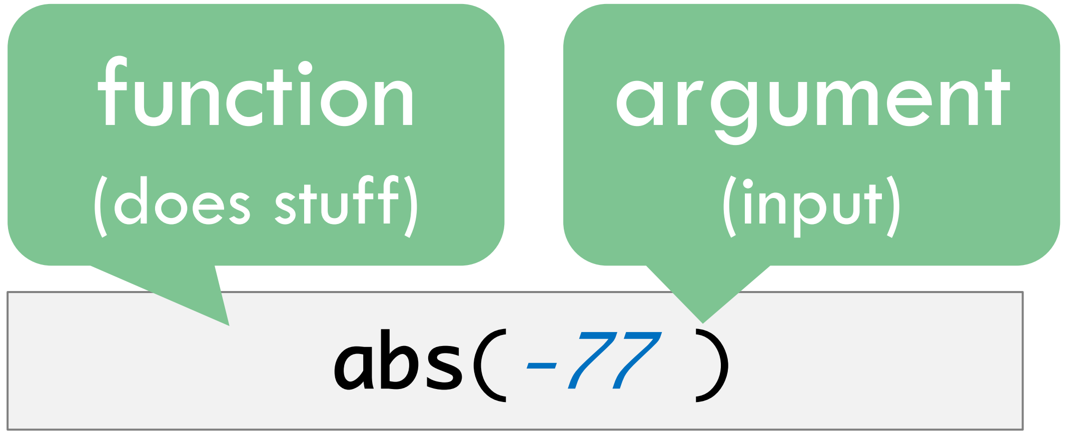
The Basics of Coding: Functions

- Code that extends our reach beyond the basic operators



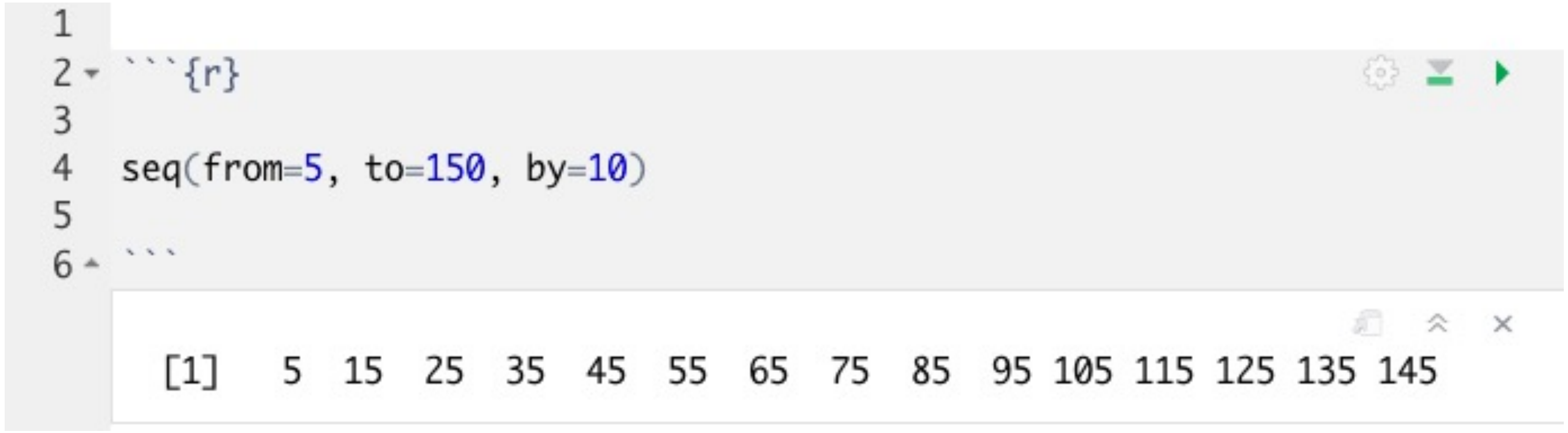
~~~
```{r}

abs(-77)

```
~~~

```
[1] 77
```



Putting Functions to Work

- We can use functions to do more than simple math, we can make things!
- We can create a series of integers (a vector) using the seq() function

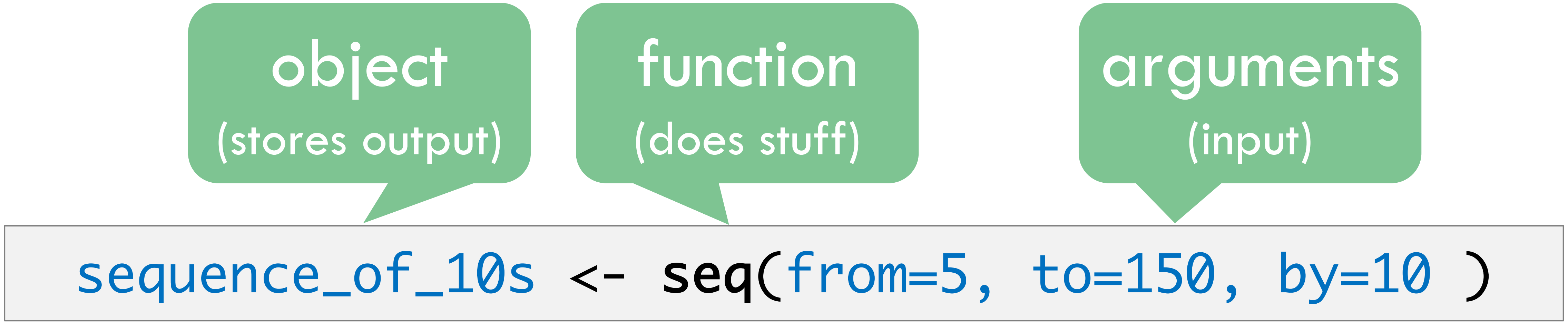


~~~
```{r}

seq(from=5, to=150, by=10)

```
~~~

```
[1]   5  15  25  35  45  55  65  75  85  95 105 115 125 135 145
```

The Basics of Coding: Objects

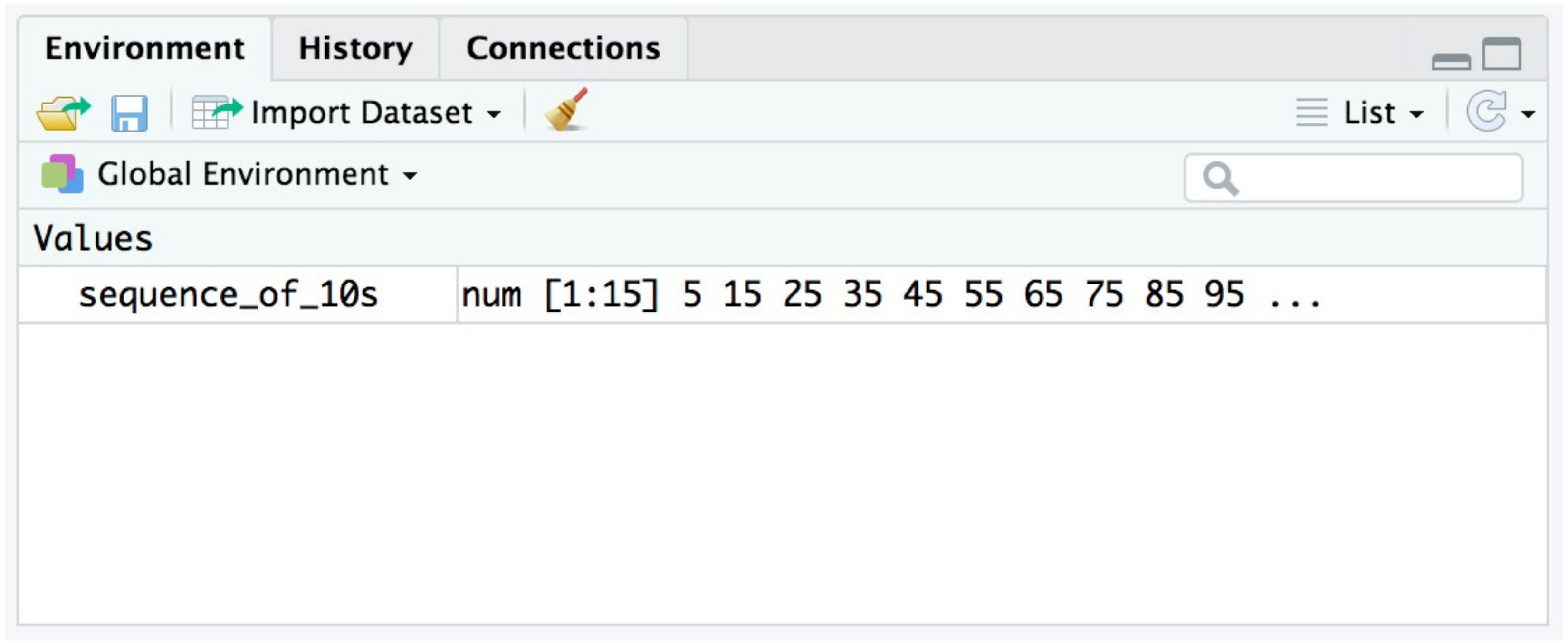
- Objects are the container for your output



```
sequence_of_10s <- seq(from=5, to=150, by=10 )
```

Checking the contents of an object

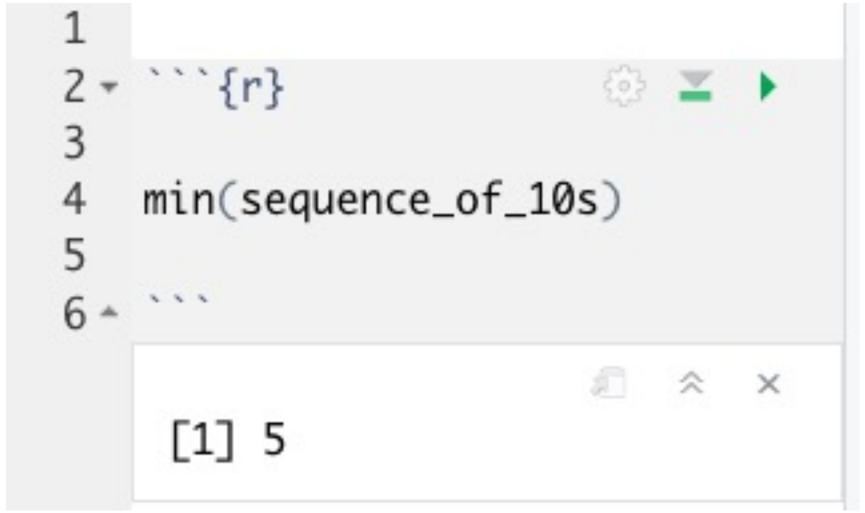
- The environment tab shows us the objects we have created.



```
Values
   sequence_of_10s    num [1:15] 5 15 25 35 45 55 65 75 85 95 ...
```

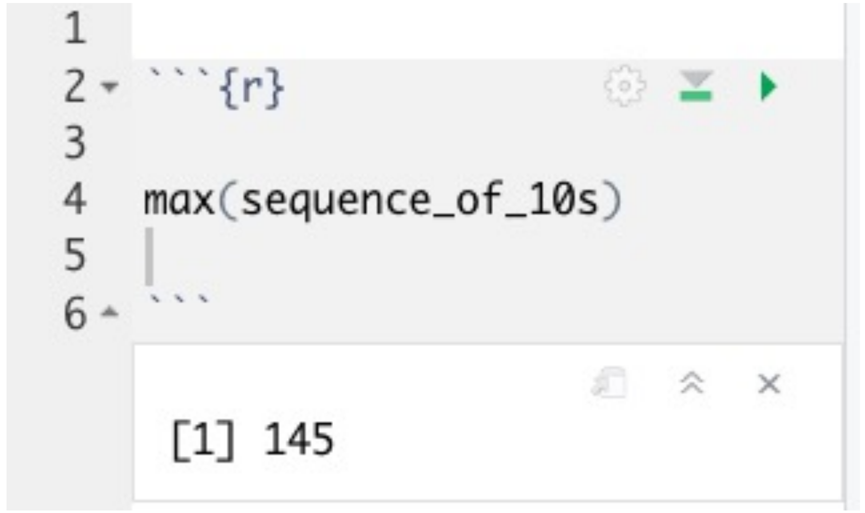
Bending objects to your will

- Once we have created an object we can start to interact with it.
- This includes passing our objects to other functions… Whoa!



~~~
```{r}

min(sequence_of_10s)

```
~~~

```
[1] 5
```

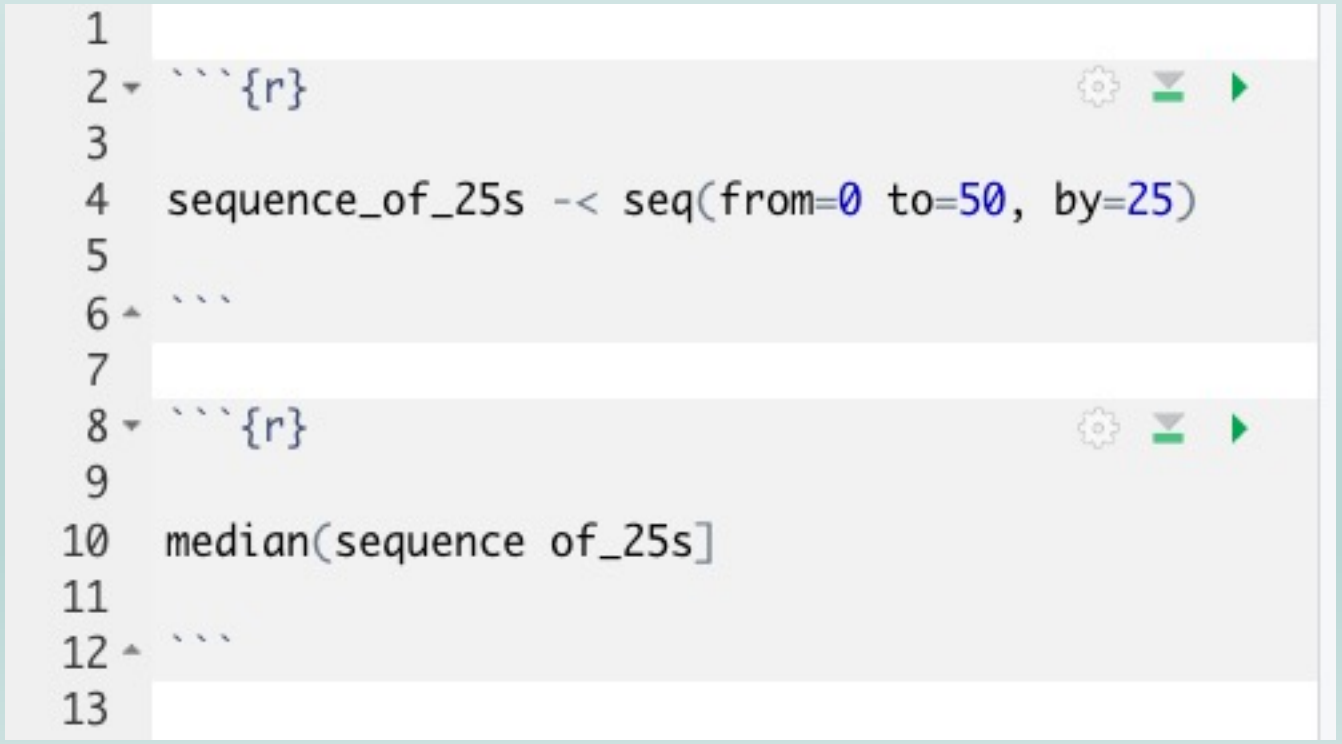


~~~
```{r}

max(sequence_of_10s)

```
~~~

```
[1] 145
```

Your Turn #1

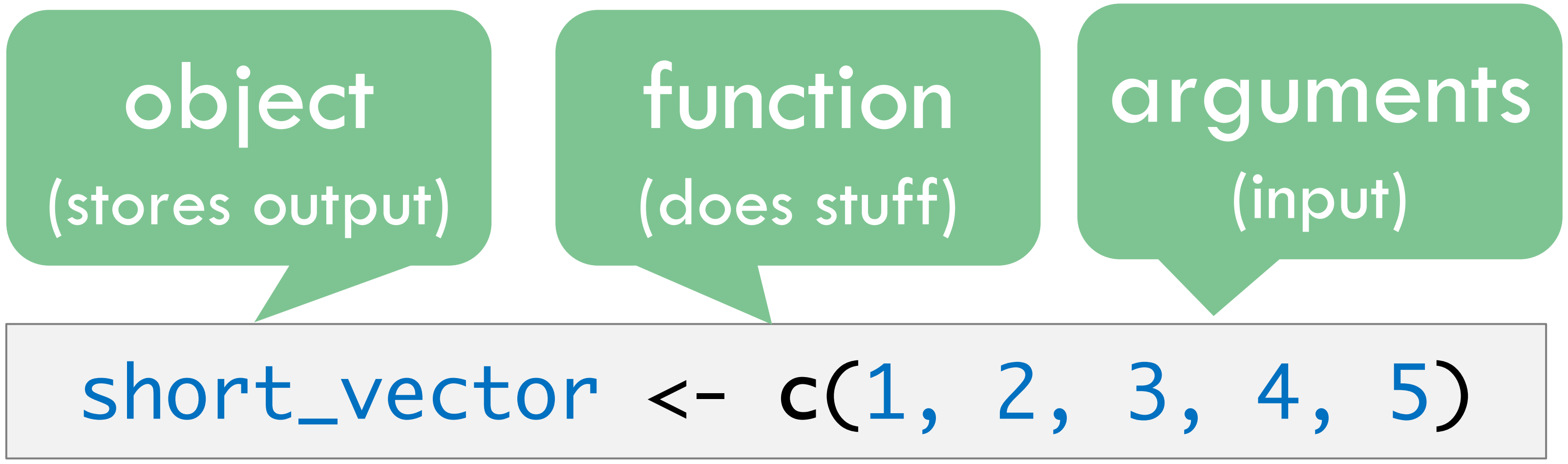
I've written some code to create a sequence from 0 to 500 in increments of 25 called sequence_of_25s. Ultimately I want to calculate the median value of this sequence. Unfortunately I've made some mistakes in my code and I am hoping you can help me find them.



~~~

```{r}

sequence_of_25s -< seq(from=0 to=50, by=25)

```

```{r}

median(sequence of_25s]

```

~~~

01:00

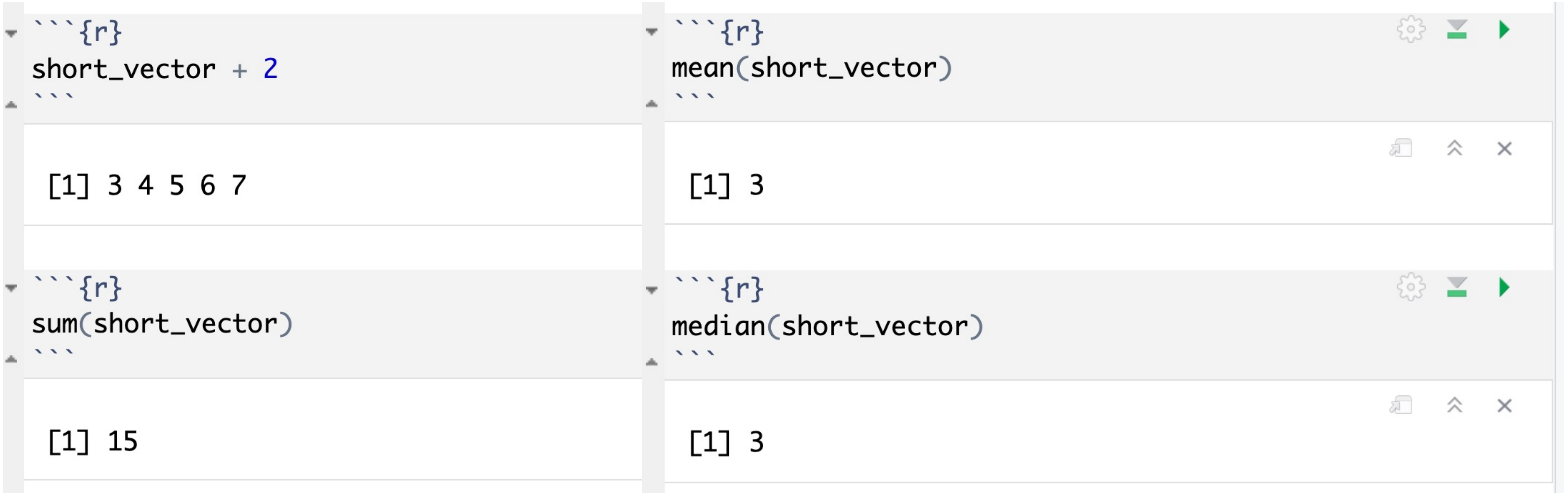
Common functions: combine values into a vector



```r
short_vector <- c(1, 2, 3, 4, 5)
```

- Create an object called short_vector and store a vector containing 1 through 5
- Think of this vector as a column of numbers you can apply functions/perform calculations on

Common functions: simple statistics



```{r}
short_vector + 2
```

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

```{r}
mean(short_vector)
```

```
[1] 3
```

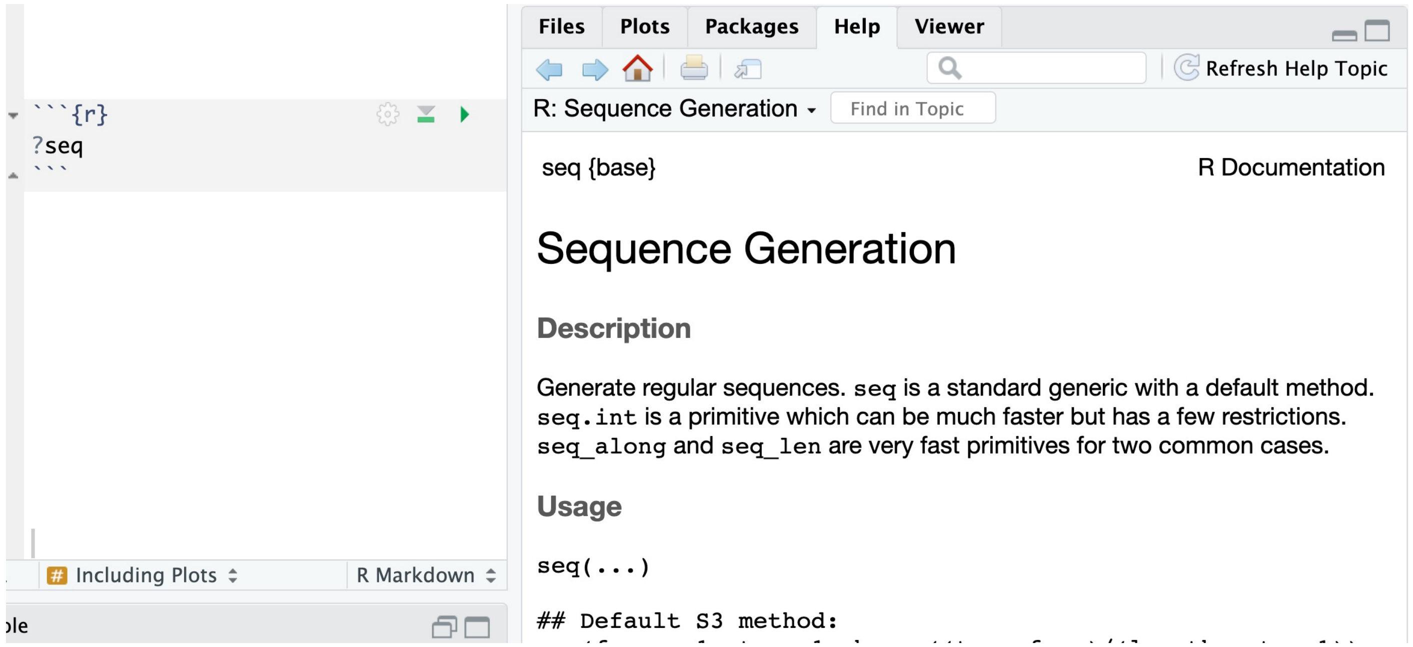
```{r}
sum(short_vector)
```

```
[1] 15
```

```{r}
median(short_vector)
```

```
[1] 3
```

How do I find out more about functions?



~~~
```{r}
?seq
```
~~~

seq {base} R Documentation

Sequence Generation

Description

Generate regular sequences. `seq` is a standard generic with a default method. `seq.int` is a primitive which can be much faster but has a few restrictions. `seq_along` and `seq_len` are very fast primitives for two common cases.

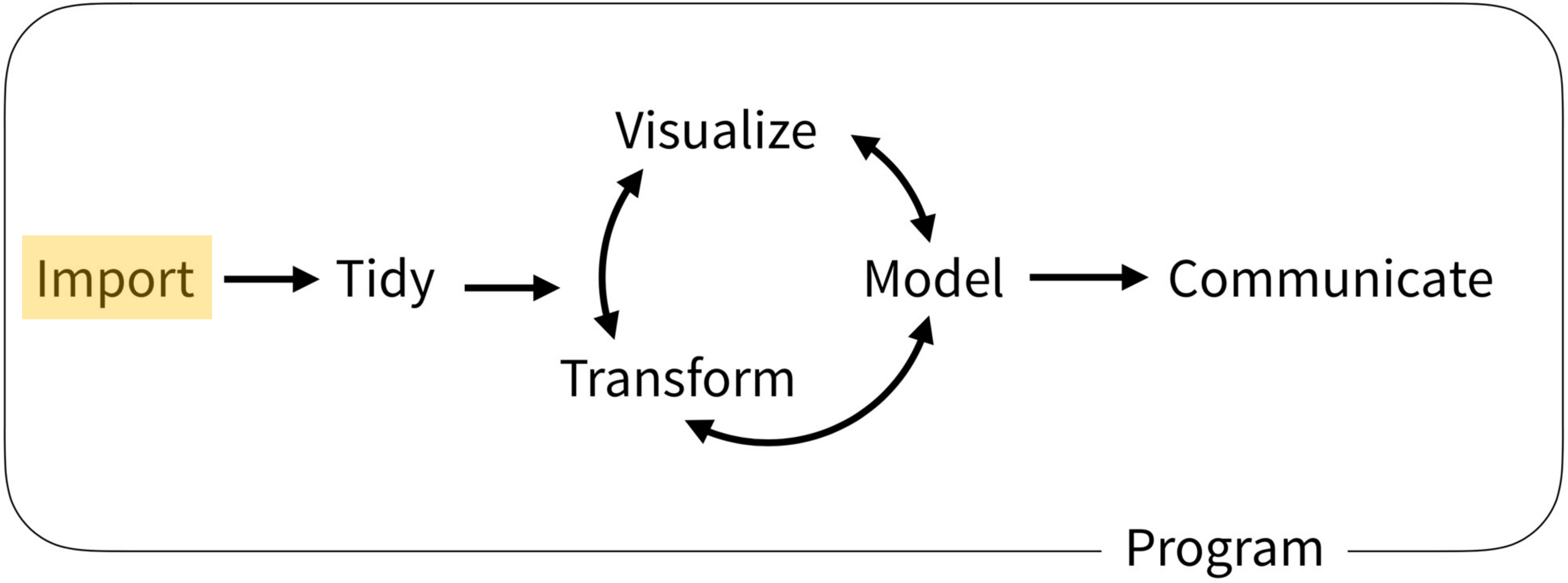
Usage

```
seq(...)

## Default S3 method:
```

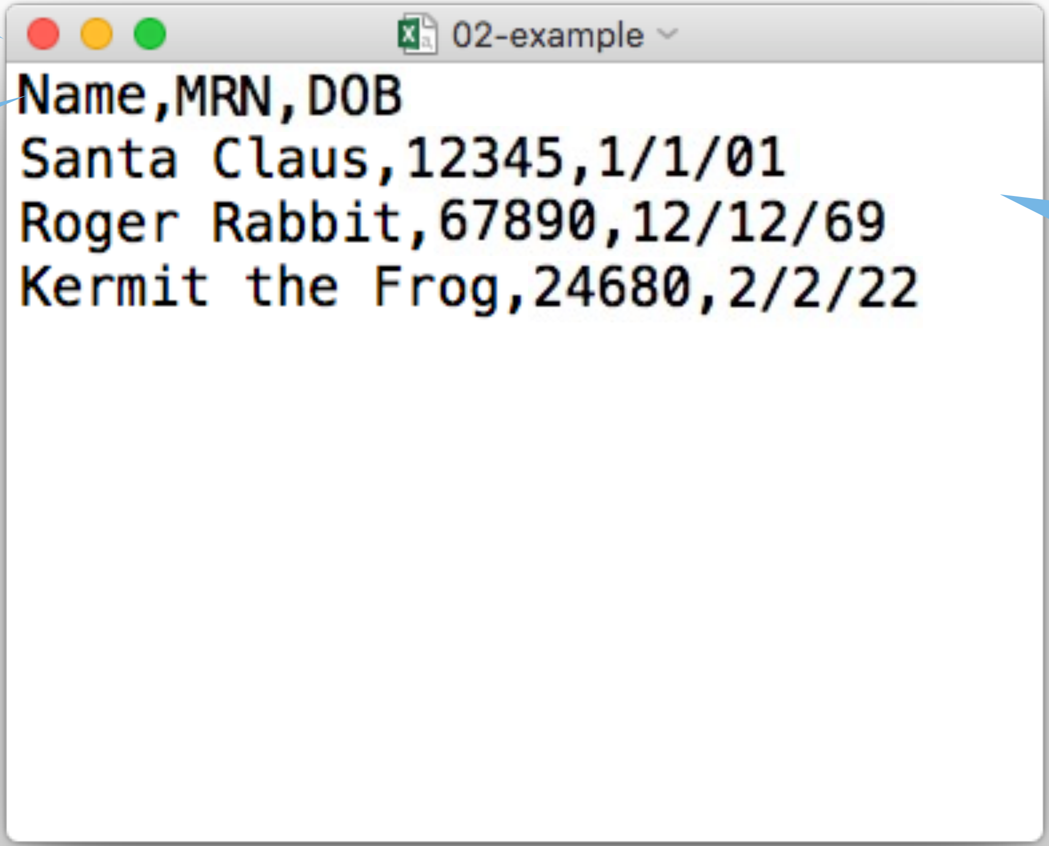
- Work Aids (RStudio Cheat Sheets: https://www.rstudio.com/resources/cheatsheets/)
- A Good Book (R for Data Science: http://r4ds.had.co.nz/)
- The Internet (Stack Overflow: https://stackoverflow.com/)

Importing Data

The Data Analysis Pipeline



From *R for Data Science* (https://r4ds.had.co.nz/introduction.html)

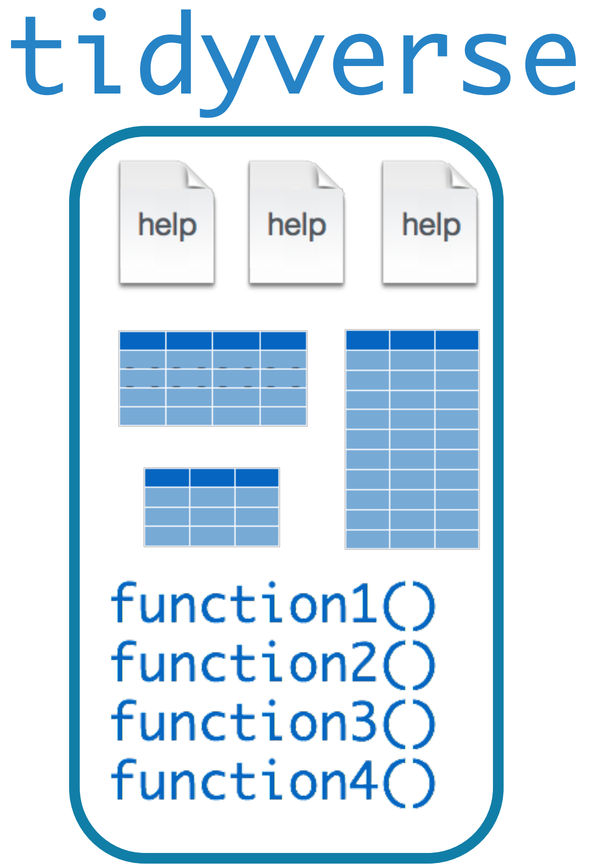


```
Name,MRN,DOB
Santa Claus,12345,1/1/01
Roger Rabbit,67890,12/12/69
Kermit the Frog,24680,2/2/22
```

Tidyverse: R Packages for Data Science

- A consistent way to organize data
- Human readable, concise, consistent code
- Build pipelines from atomic data analysis steps

Installing and loading R packages



```
install.packages("tidyverse")
```

Downloads files to computer

1 x per computer

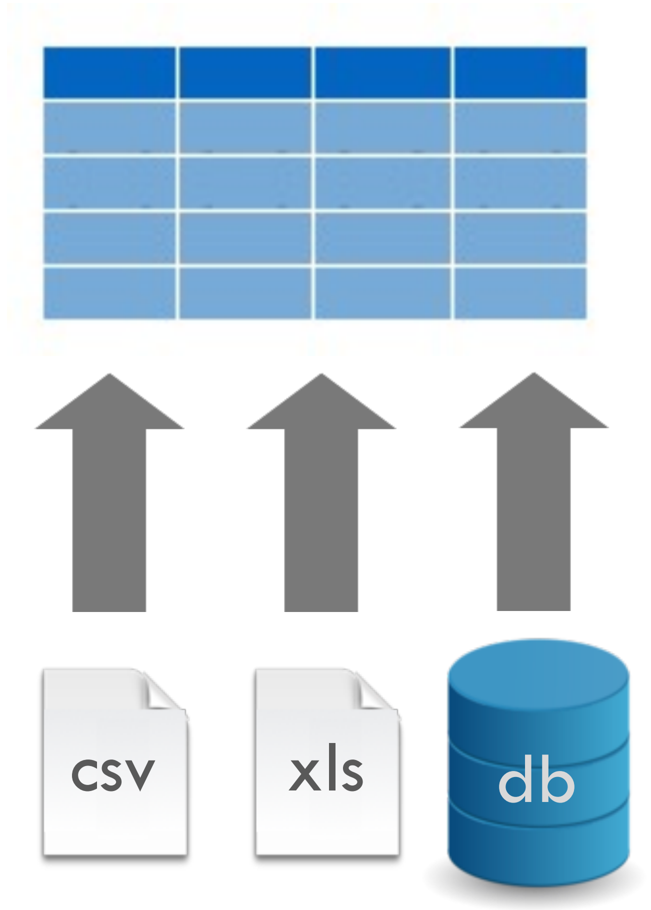
```
library("tidyverse")
```

Loads package

1 x per R Session

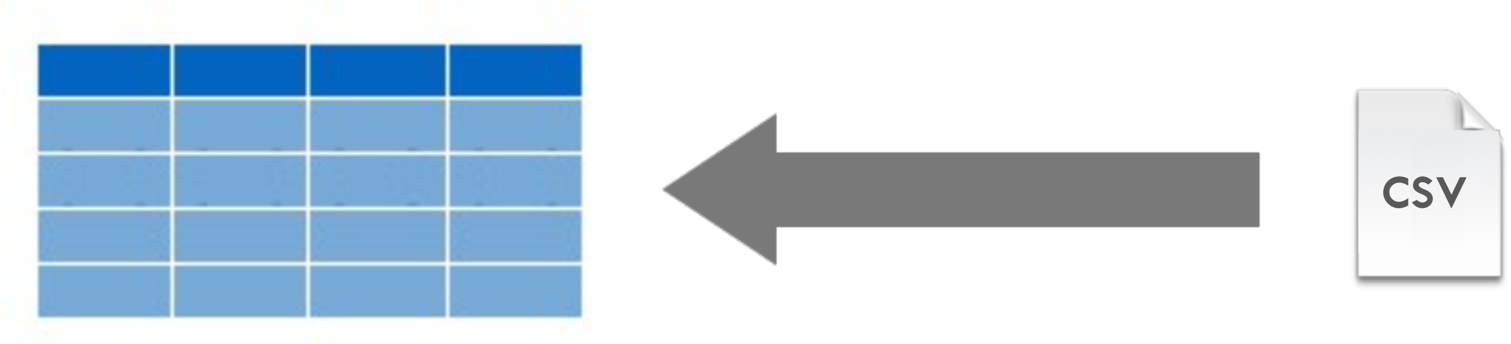
Dataframes: Beyond the Vector

- Dataframe is the term for a table
- Dataframes are composed:
  Columns (Variables)
  Rows (Observations)
- Dataframes are objects and can be acted on like other objects



read_csv()

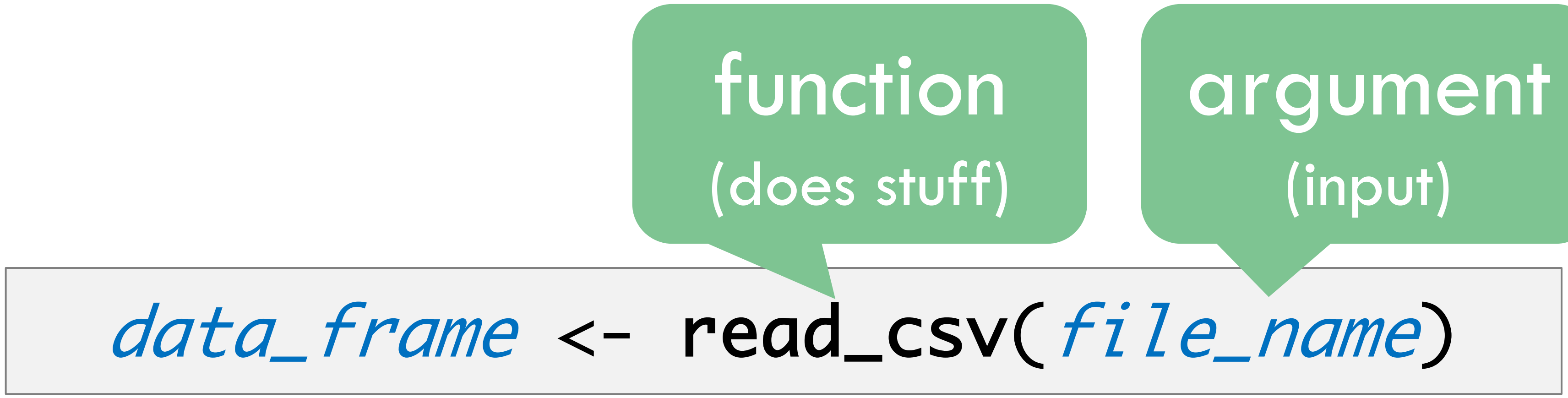
```
data_frame <- read_csv(file_name)
```



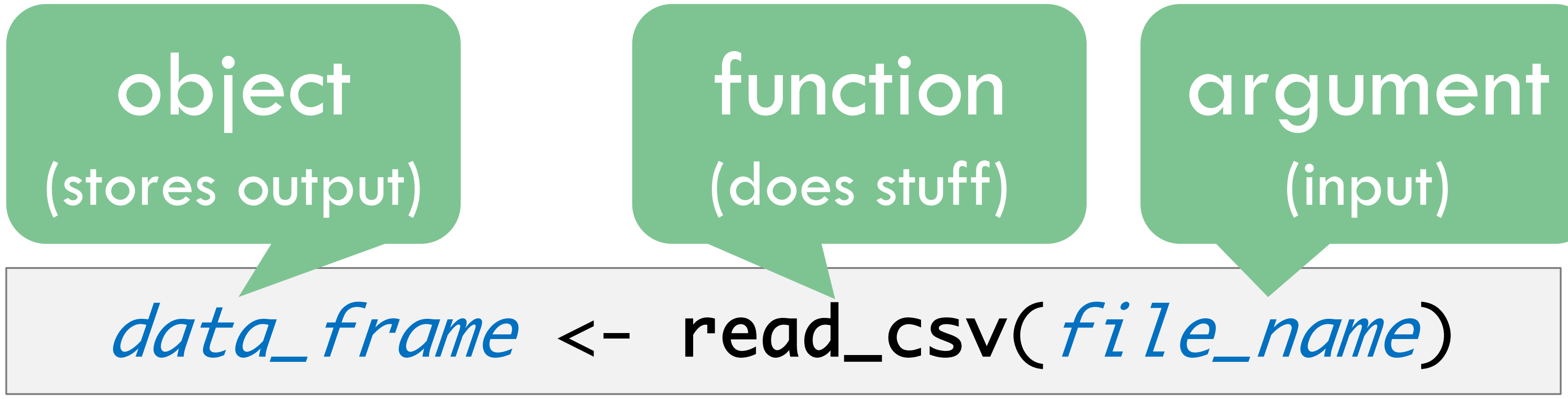
function

(does stuff)

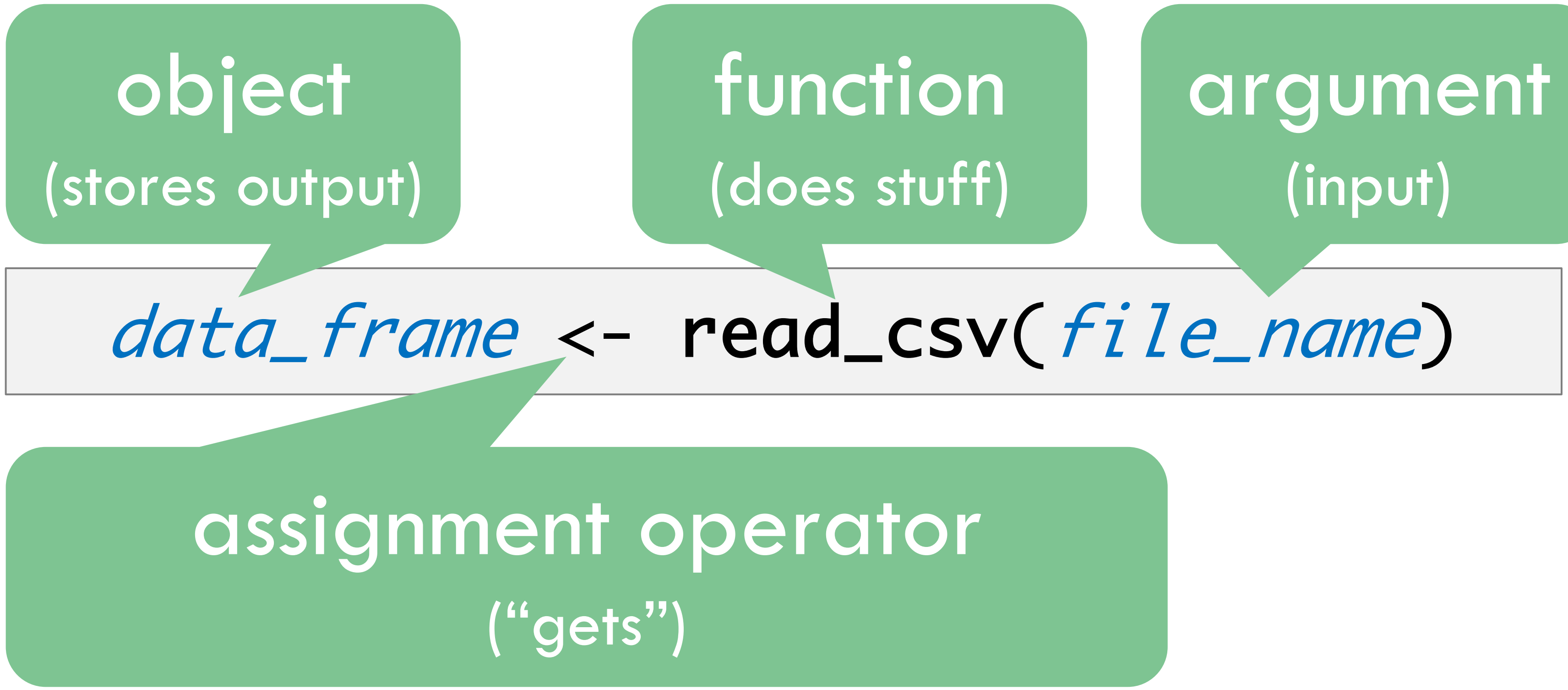
```
data_frame <- read_csv(file_name)
```



```
data_frame <- read_csv(file_name)
```



```
data_frame <- read_csv(file_name)
```



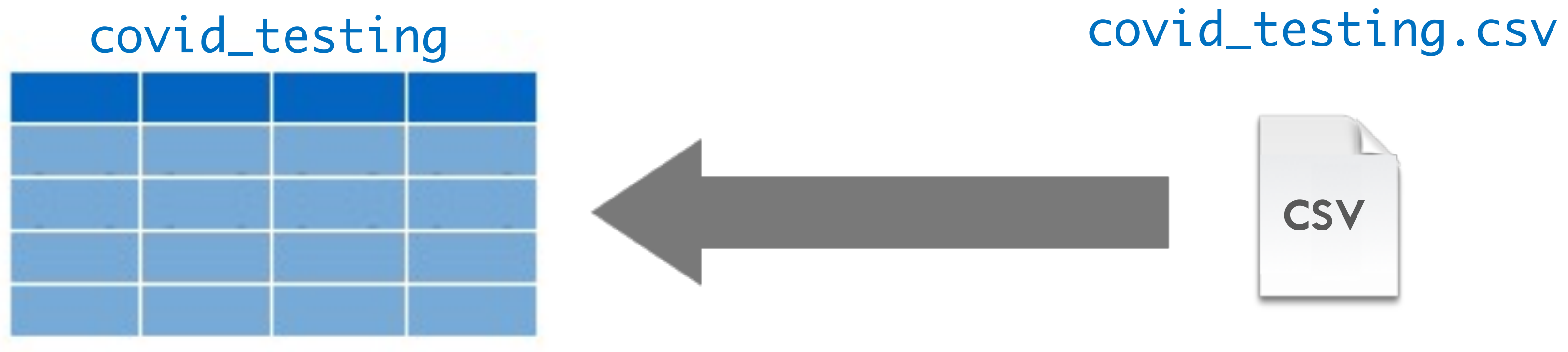
```
data_frame <- read_csv(file_name)
```

read_csv()

data frame to read data into

name of CSV file

```
covid_testing <- read_csv("covid_testing.csv")
```



Your Turn #2

In the MISC pane, select the folder:

"exercises"

Select the R Markdown file:

"`02 - Importing and Exploring Data.Rmd`"

In the Editor pane, follow the instructions to complete the exercise.

05:00

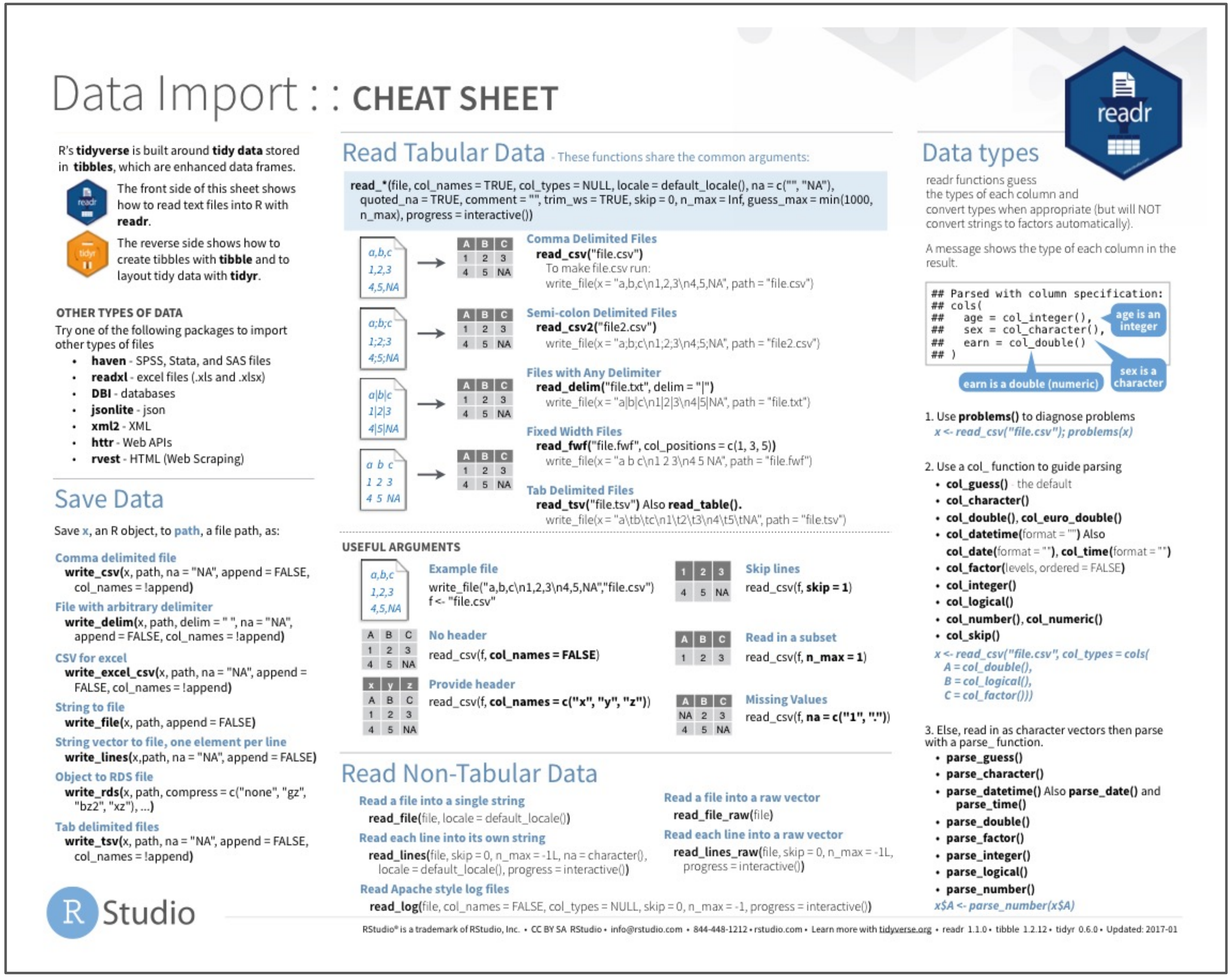
Recap

Functions do stuff. They accept **Arguments** to define parameters. We can store the output of functions in **Objects** using the assignment operator (<-).

Packages extend the functionality of R. They need to be installed once per computer and loaded each session.

Importing Data is the first step data analysis pipeline. read_csv() is a function from the tidyverse that we can use for importing data.

What else?



Data Import : : CHEAT SHEET

readr

R's **tidyverse** is built around **tidy data** stored in **tibbles**, which are enhanced data frames.

The front side of this sheet shows how to read text files into R with **readr**.

The reverse side shows how to create tibbles with **tibble** and to layout tidy data with **tidyr**.

OTHER TYPES OF DATA

Try one of the following packages to import other types of files

- **haven** - SPSS, Stata, and SAS files
- **readxl** - excel files (.xls and .xlsx)
- **DBI** - databases
- **jsonlite** - json
- **xml2** - XML
- **httr** - Web APIs
- **rvest** - HTML (Web Scraping)

Save Data

Save x, an R object, to path, a file path, as:

Comma delimited file
write_csv(x, path, na = "NA", append = FALSE, col_names = !append**)**

File with arbitrary delimiter
write_delim(x, path, delim = " ", na = "NA", append = FALSE, col_names = !append**)**

CSV for excel
write_excel_csv(x, path, na = "NA", append = FALSE, col_names = !append**)**

String to file
write_file(x, path, append = FALSE**)**

String vector to file, one element per line
write_lines(x,path, na = "NA", append = FALSE**)**

Object to RDS file
write_rds(x, path, compress = c("none", "gz", "bz2", "xz"), ...**)**

Tab delimited files
write_tsv(x, path, na = "NA", append = FALSE, col_names = !append**)**

RStudio

Read Tabular Data - These functions share the common arguments:

read_*(file, col_names = TRUE, col_types = NULL, locale = default_locale(), na = c("", "NA"), quoted_na = TRUE, comment = "", trim_ws = TRUE, skip = 0, n_max = Inf, guess_max = min(1000, n_max), progress = interactive())

```
a,b,c
1,2,3
4,5,NA
```
→

A	B	C
1	2	3
4	5	NA

Comma Delimited Files
read_csv("file.csv"**)**
To make file.csv run:
write_file(x = "a,b,c\n1,2,3\n4,5,NA", path = "file.csv")

```
a;b;c
1;2;3
4;5;NA
```
→

A	B	C
1	2	3
4	5	NA

Semi-colon Delimited Files
read_csv2("file2.csv"**)**
write_file(x = "a;b;c\n1;2;3\n4;5;NA", path = "file2.csv")

```
a|b|c
1|2|3
4|5|NA
```
→

A	B	C
1	2	3
4	5	NA

Files with Any Delimiter
read_delim("file.txt", delim = "|"**)**
write_file(x = "a|b|c\n1|2|3\n4|5|NA", path = "file.txt")

```
a b c
1 2 3
4 5 NA
```
→

A	B	C
1	2	3
4	5	NA

Fixed Width Files
read_fwf("file.fwf", col_positions = c(1, 3, 5)**)**
write_file(x = "a b c\n1 2 3\n4 5 NA", path = "file.fwf")

Tab Delimited Files
read_tsv("file.tsv"**)** Also **read_table().**
write_file(x = "a\tb\tc\n1\t2\t3\n4\t5\tNA", path = "file.tsv")

USEFUL ARGUMENTS

```
a,b,c
1,2,3
4,5,NA
```

Example file
write_file("a,b,c\n1,2,3\n4,5,NA","file.csv")
f <- "file.csv"

A	B	C
1	2	3
4	5	NA

No header
read_csv(f, **col_names = FALSE**)

x	y	z
A	B	C
1	2	3
4	5	NA

Provide header
read_csv(f, **col_names = c("x", "y", "z")**)

1	2	3
4	5	NA

Skip lines
read_csv(f, **skip = 1**)

A	B	C
1	2	3

Read in a subset
read_csv(f, **n_max = 1**)

A	B	C
NA	2	3
4	5	NA

Missing Values
read_csv(f, **na = c("1", ".")**)

Read Non-Tabular Data

Read a file into a single string
read_file(file, locale = default_locale()**)**

Read each line into its own string
read_lines(file, skip = 0, n_max = -1L, na = character(), locale = default_locale(), progress = interactive()**)**

Read Apache style log files
read_log(file, col_names = FALSE, col_types = NULL, skip = 0, n_max = -1, progress = interactive()**)**

Read a file into a raw vector
read_file_raw(file**)**

Read each line into a raw vector
read_lines_raw(file, skip = 0, n_max = -1L, progress = interactive()**)**

Data types

readr functions guess the types of each column and convert types when appropriate (but will NOT convert strings to factors automatically).

A message shows the type of each column in the result.

```
## Parsed with column specification:
## cols(
##   age = col_integer(),
##   sex = col_character(),
##   earn = col_double()
## )
```

1. Use **problems()** to diagnose problems
   x <- read_csv("file.csv"); problems(x)

2. Use a col_ function to guide parsing
   - **col_guess()** - the default
   - **col_character()**
   - **col_double()**, **col_euro_double()**
   - **col_datetime(**format = ""**)** Also **col_date(**format = ""**)**, **col_time(**format = ""**)**
   - **col_factor(**levels, ordered = FALSE**)**
   - **col_integer()**
   - **col_logical()**
   - **col_number()**, **col_numeric()**
   - **col_skip()**

   *x <- read_csv("file.csv", col_types = cols(
   A = col_double(),
   B = col_logical(),
   C = col_factor()))*

3. Else, read in as character vectors then parse with a parse_ function.
   - **parse_guess()**
   - **parse_character()**
   - **parse_datetime()** Also **parse_date()** and **parse_time()**
   - **parse_double()**
   - **parse_factor()**
   - **parse_integer()**
   - **parse_logical()**
   - **parse_number()**

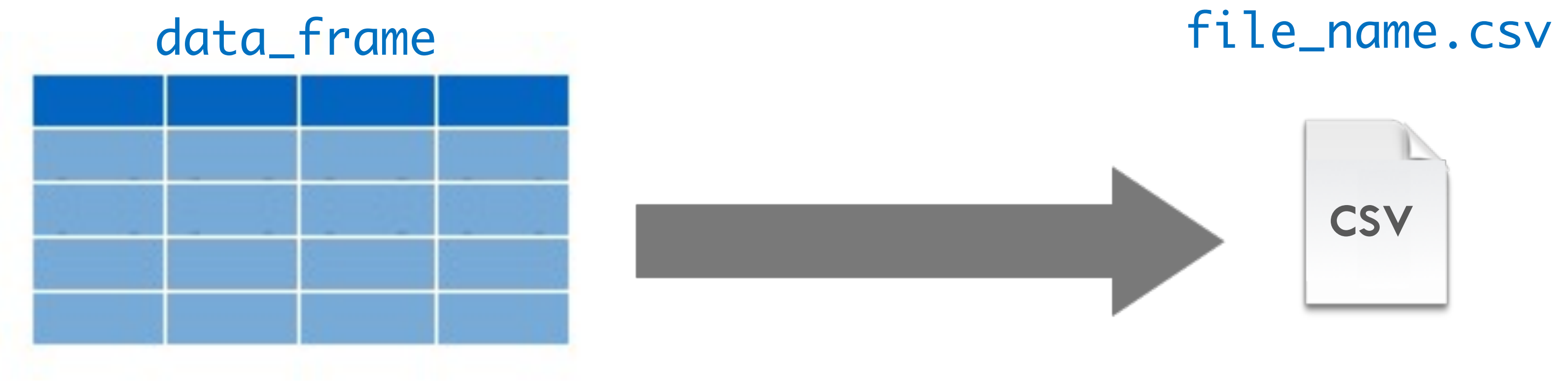
   x$A <- parse_number(x$A)

Write data to files

data frame to write file from

name of CSV file to write to

```
write_csv(data_frame, “file_name.csv”)
```



readxl

www.rstudio.com

googlesheets4

www.rstudio.com

{ jsonlite }

haven

www.rstudio.com

rvest

www.rstudio.com

Reading Excel files

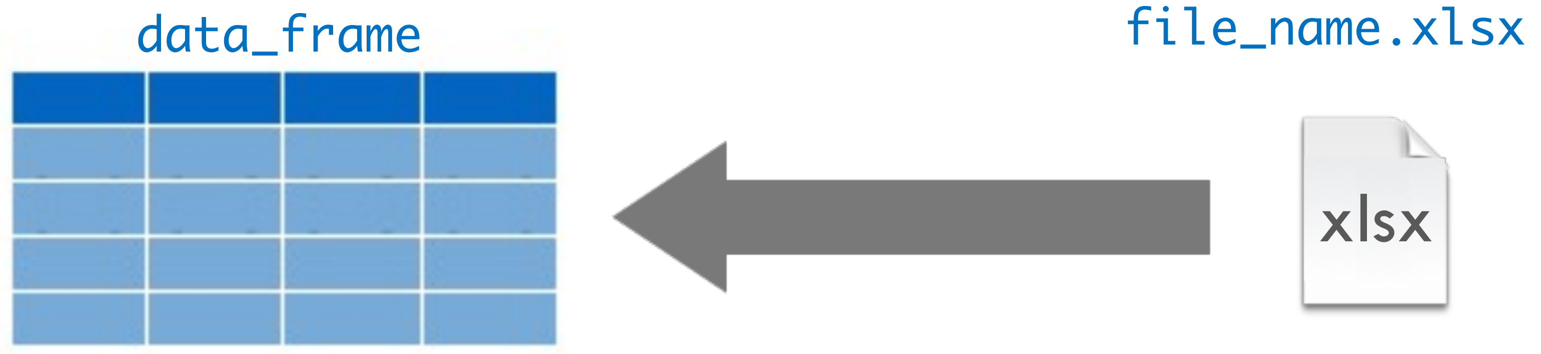
Load package

```
library(readxl)
```

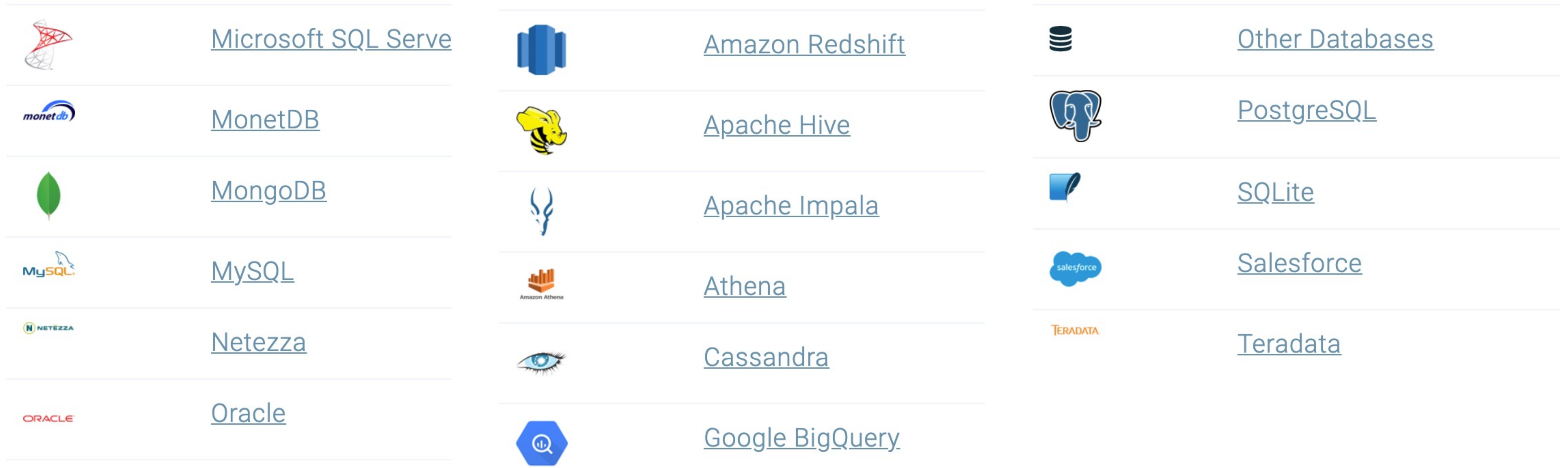
data frame to read data into

name of CSV file

```
data_frame <- read_excel(”file_name.xlsx")
```



Databases



- Microsoft SQL Serve
- MonetDB
- MongoDB
- MySQL
- Netezza
- Oracle
- Amazon Redshift
- Apache Hive
- Apache Impala
- Athena
- Cassandra
- Google BigQuery
- Other Databases
- PostgreSQL
- SQLite
- Salesforce
- Teradata

https://db.rstudio.com/databases/

R Interface to Python

```r
library(reticulate)
```

~~~~
```{python}
import pandas
covid_testing.info()
```
~~~~

Evaluation

https://forms.office.com/r/ntgSTKfvrv