A Handbook on R

Freddy Ray Drennan

2021-11-26

A Tour of R

It's all about functions

I think understanding what we are able to do with functions will help you understand how to solve the problems you are going to face. What we really care about is the ability to port around complex tools that make our tasks simpler. Functions are often described as boxes. These boxes take inputs and create outputs and can do things such as

- Access/build APIs and databases, store data locally or remote.
- Build models and make predictions based on the data we collect.
- Edit and manipulate our data into new, more useful forms
- Display our data on the internet in the form on web based applications.

Function Names

Functions can be named or unnamed. Unnamed functions are called anonymous functions. Anonymous functions are generally used when you have a simple but unique task that is applied to multiple objects sequentially, but which also is unlikely to be repeated anywhere else. For example, below we create a function which takes a value `x` and returns `x+2`. `sapply` is a special kind of function called a mapping function. It takes something (the first argument) an `X` and applies a function `FUN` to each element of `X`. In this case, the numbers 1 to 10.

```
add_two <- function(x) x + 2
sapply(X = 1:10, FUN = add_two)
```

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

```
sapply(X = 1:10, FUN = function(x) x + 2)
```

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

The input is whatever is contained within the parenthesis `()` and the output will be displayed below the code. We will talk more about functions later, so don't try to get too caught up in the details here. Programming is something I want you to soak in, not memorize.

Function Theory

The following unnamed function, a special case called an anonymous function or lambda function, $(x, y) \to x(x+y)$ has two critical components. Don't be afraid of the right arrow. We just say that x and y map *onto* whatever expression is on the right hand side.

1. The function has any number of inputs, in the case above there are two: (x, y)
2. The function has a body which defines the output, $x(x + y)$
3. We can express evaluation of the function by following the function with inputs:

- $((x, y) \to x(x + y))(1, 2)$ which evaluates to $(1, 2) \to 1 * (1 + 2) = 3$

When we write functions in R we will use both named and unnamed functions. But lets start with the first example. This function is an anonymous function. We haven't given it a name yet. But we can call it all the same.

```
function(x, y) x * (x + y)
```

```
## function(x, y) x * (x + y)
```

```
# Wrapping function in () means evaluate now
(function(x, y) x * (x + y))(1, 2)
```

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

Now wouldn't it be nice if we could take this function with us? Let's give it a name and update our notation $xy\_calc(x, y) \to x(x + y)$

```
library(cli)

xy_calc <- function(x, y) {
  x_name <- deparse1(substitute(x))
```

```
  y_name <- deparse1(substitute(y))
  cli_alert_info('{x_name} is {x} and {y_name} is {y}')
  result <- x * (x + y)
  cli_alert_success('result is {result}')
  result
}

new_calc <- function(a, b) xy_calc(xy_calc(a, a), xy_calc(b, b))
```

Now we can describe silly things like this $newcalc(x, y) \rightarrow xy\_calc(xy\_calc(x, x), xy\_calc(y, y)))$

```
new_calc(1, 2)
```

```
## i a is 1 and a is 1

## v result is 2

## i b is 2 and b is 2

## v result is 8

## i xy_calc(a, a) is 2 and xy_calc(b, b) is 8

## v result is 20

## [1] 20
```

Vectors and Lists

The `c` function creates something called a vector, which is a 1 dimensional matrix or table. Vectors are the singular columns or rows in a table. They do not mix types. The types you will work with are primarily logical, integer, double, character, list, NULL, closure (function).

There are two functions that we should get familiar with first, but let's go with the one I think you will use more of the two, though both `list` and `c` will be functions you use daily.

```
c(integer = 1L, double = 1, bool = TRUE, character = 'a')
```

```
##   integer    double      bool character
##       "1"       "1"    "TRUE"       "a"
```

```
list(integer = 1L, double = 1, bool = TRUE, character = 'a')
```

```
## $integer
## [1] 1
##
## $double
## [1] 1
##
## $bool
## [1] TRUE
##
## $character
## [1] "a"
```

data.frame and tibble

The table below has 336,776 rows. Get used to looking at data in a different way. We're not in Excel anymore and sometimes it makes people feel funny about not being able to "see" their data. When I say see, I simply mean is that look at the data below. This is a two dimensional table called a dataframe or tibble. A dataframe is a special form of a list, that requires each element be atomic (of one type) and of equal length. We can see this is true by looking at the output below.

```
## Rows: 336,776
## Columns: 19
## $ year           <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2~
## $ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
## $ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
## $ dep_time       <int> 517, 533, 542, 544, 554, 554, 555, 557, 557, 558, 558, ~
## $ sched_dep_time <int> 515, 529, 540, 545, 600, 558, 600, 600, 600, 600, 600, ~
## $ dep_delay      <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3, -2, -2, -2, -2, -2, -1~
## $ arr_time       <int> 830, 850, 923, 1004, 812, 740, 913, 709, 838, 753, 849,~
## $ sched_arr_time <int> 819, 830, 850, 1022, 837, 728, 854, 723, 846, 745, 851,~
## $ arr_delay      <dbl> 11, 20, 33, -18, -25, 12, 19, -14, -8, 8, -2, -3, 7, -1~
## $ carrier        <chr> "UA", "UA", "AA", "B6", "DL", "UA", "B6", "EV", "B6", "~
## $ flight         <int> 1545, 1714, 1141, 725, 461, 1696, 507, 5708, 79, 301, 4~
## $ tailnum        <chr> "N14228", "N24211", "N619AA", "N804JB", "N668DN", "N394~
## $ origin         <chr> "EWR", "LGA", "JFK", "JFK", "LGA", "EWR", "EWR", "LGA",~
## $ dest           <chr> "IAH", "IAH", "MIA", "BQN", "ATL", "ORD", "FLL", "IAD",~
## $ air_time       <dbl> 227, 227, 160, 183, 116, 150, 158, 53, 140, 138, 149, 1~
## $ distance       <dbl> 1400, 1416, 1089, 1576, 762, 719, 1065, 229, 944, 733, ~
## $ hour           <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, 6~
## $ minute         <dbl> 15, 29, 40, 45, 0, 58, 0, 0, 0, 0, 0, 0, 0, 0, 0, 59, 0~
```

```
## $ time_hour      <dttm> 2013-01-01 05:00:00, 2013-01-01 05:00:00, 2013-01-01 0~
```

There are $19 * 336,776$ cells within this table because there are 19 vectors (columns) of length $336,776$. Each element in the vector is one instance of a string, a number, a date, etc. And each type has functions that can operate on it, in a repeatable way. This is why each column (vector) is of one type, either `int`, `dbl`, `chr`, or `dttm`. There are other types in R, but we will be sticking to these for now. `int` is a special form of number called an integer - 1, 2, -1231, etc. `dbl` stands for double, a number which can contain decimal values i.e, 1.3. Not everything needs to be double, because an integer takes less memory in your computer to store. Don't get too caught up on whether or not something should be an integer or double and generally speaking you wont consciously make a choice.

If we were to write by hand the expression we created, it would be

```
library(cli)

# Same as f(x) = 2x + 3
f <-  function(x) {
  x <- x * x
  x
}

g <- function(x=NULL, y=NULL) {
  result <- x + min(x, y)
  result
}

print(f(g(3, 4)))
```

```
## [1] 36
```

We now have a way of describing inputs and output a little more clearly. Instead of writing, $(3 + min(3,4)) * (3 + min(3,4))$ we can write $f(g(3,4))$ or try new creations like $z(x,y) = f(g(f(x), f(y)))$ so $z(1,2) = f(g(f(1), f(2))) = f(g(1,4)) = f(2) = 4$.

Now just take this idea about functions and expand your definition of inputs and outputs to be any number, none or many, and of any type that R supports - character, numeric, date/time, data.frame or list - all of which we'll cover.

Solve a Problem in R

Let's solve a problem using R. Suppose we have a friend that is interested in the current trend regarding COVID-19 cases. The first thing we will probably do is try to figure out an efficient and reliable way for importing Covid-19 data into our R session. Conveniently, the `COVID19` package allows us to pull the latest data without any hard work and consists of one function - `covid19`.

Installing Packages

In your RStudio console, you can write the following to install the `COVID19` package using the `install.packages` function. If you are interested in learning more about this function, you can write `?install.packages` in your console and the documentation for the function will appear.

```
# For help menu, uncomment next line
# ?install.packages

# If the package is not yet installed, you can install it by passing
# a string with the package name to the `install.packages` function
install.packages(pkgs = c('COVID19'))
```

Available Packages on CRAN

For a full list of what packages are available through the `install.packages` function, please check out the Contributed Packages page at CRAN or scrape it yourself.

```
library(rvest)
cran_packages <- 'https://cran.r-project.org/web/packages/available_packages_by_date.ht
package_data <- html_table(html_element(read_html(cran_packages), 'table'))
print(package_data)
```

```
## # A tibble: 18,500 x 3
##    Date       Package         Title
##    <chr>      <chr>           <chr>
##  1 2021-11-25 aMNLFA          Automated Moderated Nonlinear Factor Analysis Usi~
##  2 2021-11-25 audio           Audio Interface for R
##  3 2021-11-25 boot.pval       Bootstrap p-Values
##  4 2021-11-25 bootUR          Bootstrap Unit Root Tests
##  5 2021-11-25 CALIBERrfimpute Multiple Imputation Using MICE and Random Forest
##  6 2021-11-25 filearray       File-Backed Array for Out-of-Memory Computation
```

```
##  7 2021-11-25 gamlss.foreach  Parallel Computations for Distributional Regressi~
##  8 2021-11-25 ggquiver        Quiver Plots for 'ggplot2'
##  9 2021-11-25 ICSKAT          Interval-Censored Sequence Kernel Association Test
## 10 2021-11-25 mapscanner      Print Maps, Draw on Them, Scan Them Back in
## # ... with 18,490 more rows
```

```
n_packages <- length(unique(package_data$Package))
cli_alert_info('There are {n_packages} packages on CRAN')
```

```
## i There are 18500 packages on CRAN
```

Using Functions to Solve a Problem

The code below consists of three different functions. The first two are `library` and `covid19`, but the third is hidden - it's actually the arrow, `<-` if you execute `` `<-`(a, 1) `` the output of the function actually creates the variable `a` within your session! *Functions* are spaces for stuff to happen. Functions help us make common procedures repeatable. By creating a function with a particular name and inputs, we can get some sort of useful (or not useful, the world's your oyster) output.

In this case, `library` loads packages from a folder in the R environment called library. You can see which ones your R environment knows about by running the function `.libPaths()`. The dot in front of `.libPaths()` just means that the author intended it to be hidden, which doesn't really mean much to us. When you run `install.packages` that code is at a path in the `.libPaths()` output.

`covid19` is a function from the `COVID19` package, and would only be available after executing `library(COVID19)` or if `library(COVID19)` is omitted, by pulling it from the package namespace directly by preceding the function with the package name and two colons like so: `COVID19::covid19`. Generally speaking, you simply use `library` because it reduces the amount of text on the page.

```
library(tidyverse)
library(purrr)
library(COVID19)
```

```
covid_data <- covid19(
    country = 'United States',
    start = '2021-01-01',
    end = "2021-11-21",
    verbose = FALSE,
    level = 2
```

```
)
glimpse(covid_data)
```

```
## Rows: 18,200
## Columns: 47
## $ id                                   <chr> "10b692cc", "10b692cc", "10b692cc"~
## $ date                                 <date> 2021-01-01, 2021-01-02, 2021-01-0~
## $ confirmed                            <int> 122, 122, 122, 122, 122, 124, 125,~
## $ deaths                               <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2~
## $ recovered                            <int> 29, 29, 29, 29, 29, 29, 29, 29, 29~
## $ tests                                <int> 27102, 27132, 27143, 27419, 27525,~
## $ vaccines                             <int> 3052, 3052, 3052, 3094, 3094, 3105~
## $ people_vaccinated                    <int> 3051, 3051, 3051, 3093, 3093, 3104~
## $ people_fully_vaccinated              <int> 1, 1, 1, 1, 1, 1, 1, 1, 95, 181, 3~
## $ hosp                                 <int> NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ icu                                  <int> NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ vent                                 <int> NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ school_closing                       <int> NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ workplace_closing                    <int> NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ cancel_events                        <int> NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ gatherings_restrictions              <int> NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ transport_closing                    <int> NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ stay_home_restrictions               <int> NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ internal_movement_restrictions       <int> NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ international_movement_restrictions <int> NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ information_campaigns                <int> NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ testing_policy                       <int> NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ contact_tracing                      <int> NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ facial_coverings                     <int> NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ vaccination_policy                   <int> NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ elderly_people_protection            <int> NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ government_response_index            <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ stringency_index                     <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ containment_health_index             <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ economic_support_index               <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ administrative_area_level            <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2~
## $ administrative_area_level_1          <chr> "United States", "United States", ~
## $ administrative_area_level_2          <chr> "Northern Mariana Islands", "North~
## $ administrative_area_level_3          <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ latitude                             <dbl> 14.15569, 14.15569, 14.15569, 14.1~
## $ longitude                            <dbl> 145.2119, 145.2119, 145.2119, 145.~
## $ population                           <int> 55144, 55144, 55144, 55144, 55144,~
## $ iso_alpha_3                          <chr> "USA", "USA", "USA", "USA", "USA",~
## $ iso_alpha_2                          <chr> "US", "US", "US", "US", "US", "US"~
## $ iso_numeric                          <int> 840, 840, 840, 840, 840, 840, 840,~
```

```
## $ iso_currency                    <chr> "USD", "USD", "USD", "USD", "USD",~
## $ key_local                       <chr> "69", "69", "69", "69", "69", "69"~
## $ key_google_mobility             <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ key_apple_mobility              <chr> "Northern Mariana Islands", "North~
## $ key_jhu_csse                    <chr> "US69", "US69", "US69", "US69", "U~
## $ key_nuts                        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ key_gadm                        <chr> "MNP", "MNP", "MNP", "MNP", "MNP",~
```

Let's look at what happened - we passed a few inputs and received a dataframe. A dataframe is a list with the requirement that all elements of the list are atomic vectors of equal length. Let's look at what that means.

```
map_chr(covid_data, typeof)
```

```
##                                 id                               date
##                        "character"                           "double"
##                          confirmed                             deaths
##                          "integer"                          "integer"
##                          recovered                              tests
##                          "integer"                          "integer"
##                           vaccines                 people_vaccinated
##                          "integer"                          "integer"
##            people_fully_vaccinated                               hosp
##                          "integer"                          "integer"
##                                icu                               vent
##                          "integer"                          "integer"
##                       school_closing                 workplace_closing
##                          "integer"                          "integer"
##                       cancel_events           gatherings_restrictions
##                          "integer"                          "integer"
##                  transport_closing            stay_home_restrictions
##                          "integer"                          "integer"
##      internal_movement_restrictions international_movement_restrictions
##                          "integer"                          "integer"
##               information_campaigns                      testing_policy
##                          "integer"                          "integer"
##                     contact_tracing                   facial_coverings
##                          "integer"                          "integer"
##                  vaccination_policy         elderly_people_protection
##                          "integer"                          "integer"
##          government_response_index                   stringency_index
##                           "double"                           "double"
##            containment_health_index             economic_support_index
##                           "double"                           "double"
##          administrative_area_level       administrative_area_level_1
```

```
##                          "integer"                          "character"
##       administrative_area_level_2       administrative_area_level_3
##                          "character"                          "character"
##                             latitude                            longitude
##                             "double"                             "double"
##                           population                          iso_alpha_3
##                            "integer"                          "character"
##                           iso_alpha_2                          iso_numeric
##                           "character"                            "integer"
##                          iso_currency                            key_local
##                           "character"                          "character"
##                  key_google_mobility                  key_apple_mobility
##                           "character"                          "character"
##                          key_jhu_csse                             key_nuts
##                           "character"                            "logical"
##                              key_gadm
##                           "character"
```

When you have a list of things, you can apply a function to each item in the list. So in the list above, we have 47 atomic vectors. What does that mean? An atomic vector is like a list, but it has to contain the same thing in each cell.

How do I know if a function is vectorized
Vectorization in R

```
vector_example <- c(1, 'a', TRUE)
list_example <- list(1, 'a', TRUE)

map_chr(vector_example, typeof)
```

```
## [1] "character" "character" "character"
```

```
map_chr(list_example, typeof)
```

```
## [1] "double"    "character" "logical"
```

With the knowledge of vectors and lists, what can we do? Well, the first thing I notice is that some of the vectors are completely `NA`. Let's check the number of NA values in each vector.

```
all_na <- function(item) {
  sum(is.na(item))==length(item)
}
covid_data <- discard(covid_data, all_na)
head(covid_data)
```

```
##         id       date confirmed deaths recovered tests vaccines
## 1 10b692cc 2021-01-01       122      2        29 27102     3052
## 2 10b692cc 2021-01-02       122      2        29 27132     3052
## 3 10b692cc 2021-01-03       122      2        29 27143     3052
## 4 10b692cc 2021-01-04       122      2        29 27419     3094
## 5 10b692cc 2021-01-05       122      2        29 27525     3094
## 6 10b692cc 2021-01-06       124      2        29 27538     3105
##   people_vaccinated people_fully_vaccinated hosp icu vent school_closing
## 1              3051                       1   NA  NA   NA             NA
## 2              3051                       1   NA  NA   NA             NA
## 3              3051                       1   NA  NA   NA             NA
## 4              3093                       1   NA  NA   NA             NA
## 5              3093                       1   NA  NA   NA             NA
## 6              3104                       1   NA  NA   NA             NA
##   workplace_closing cancel_events gatherings_restrictions transport_closing
## 1                NA            NA                      NA                NA
## 2                NA            NA                      NA                NA
## 3                NA            NA                      NA                NA
## 4                NA            NA                      NA                NA
## 5                NA            NA                      NA                NA
## 6                NA            NA                      NA                NA
##   stay_home_restrictions internal_movement_restrictions
## 1                     NA                            NA
## 2                     NA                            NA
## 3                     NA                            NA
## 4                     NA                            NA
## 5                     NA                            NA
## 6                     NA                            NA
##   international_movement_restrictions information_campaigns testing_policy
## 1                                  NA                    NA             NA
## 2                                  NA                    NA             NA
## 3                                  NA                    NA             NA
## 4                                  NA                    NA             NA
## 5                                  NA                    NA             NA
## 6                                  NA                    NA             NA
##   contact_tracing facial_coverings vaccination_policy elderly_people_protection
## 1              NA               NA                 NA                        NA
## 2              NA               NA                 NA                        NA
## 3              NA               NA                 NA                        NA
## 4              NA               NA                 NA                        NA
## 5              NA               NA                 NA                        NA
## 6              NA               NA                 NA                        NA
##   government_response_index stringency_index containment_health_index
## 1                        NA               NA                       NA
## 2                        NA               NA                       NA
## 3                        NA               NA                       NA
```

```
## 4                        NA               NA                        NA
## 5                        NA               NA                        NA
## 6                        NA               NA                        NA
##   economic_support_index administrative_area_level administrative_area_level_1
## 1                     NA                        2               United States
## 2                     NA                        2               United States
## 3                     NA                        2               United States
## 4                     NA                        2               United States
## 5                     NA                        2               United States
## 6                     NA                        2               United States
##   administrative_area_level_2 latitude longitude population iso_alpha_3
## 1    Northern Mariana Islands 14.15569  145.2119      55144         USA
## 2    Northern Mariana Islands 14.15569  145.2119      55144         USA
## 3    Northern Mariana Islands 14.15569  145.2119      55144         USA
## 4    Northern Mariana Islands 14.15569  145.2119      55144         USA
## 5    Northern Mariana Islands 14.15569  145.2119      55144         USA
## 6    Northern Mariana Islands 14.15569  145.2119      55144         USA
##   iso_alpha_2 iso_numeric iso_currency key_local key_google_mobility
## 1          US         840          USD        69                <NA>
## 2          US         840          USD        69                <NA>
## 3          US         840          USD        69                <NA>
## 4          US         840          USD        69                <NA>
## 5          US         840          USD        69                <NA>
## 6          US         840          USD        69                <NA>
##         key_apple_mobility key_jhu_csse key_gadm
## 1 Northern Mariana Islands         US69      MNP
## 2 Northern Mariana Islands         US69      MNP
## 3 Northern Mariana Islands         US69      MNP
## 4 Northern Mariana Islands         US69      MNP
## 5 Northern Mariana Islands         US69      MNP
## 6 Northern Mariana Islands         US69      MNP
```

Initial Setup

Book Outline

- Install R
- Install R Studio
- Windows Only: Install RTools
  - When installed, run in the RStudio Console: `write('PATH="${RTOOLS40_HOME}\\usr\\bin;${PATH}"', file = "~/.Renviron", append = TRUE)`
- Windows Only: Install WSL2
  - Computer should be completely updated before install.
- Install Git
- Create Github Account
- Fork r-handbook
- Install Docker and Docker Compose
- Create AWS Account
  - Billing will be discussed in the course, but don't expect to pay much - i.e., 10-20 dollars a month for high course activity.
  - Remember to `stop` EC2 servers when we begin using them. AWS is polite about your first few refund requests.
- Create Reddit Account
  - Follow Instructions here

Make sure you install the `tidyverse` packages. Update to renv later.

```
install.packages('tidyverse')
```

What is R

Types of Problems You Can Solve

Base R, Tidyverse, data.table

Arguments/ Developments within the language

What are Variables

Valid Variable Names

Building Blocks

Vectors

Vectors are containers information of similar type. You can think of them as having $1 * n$ cells where n is any positive integer, and make up the rows and columns of tables. Vectors always contain the same type of value. R has many different types of vectors, but the most common are **numeric**, **character**, and **logical** (**TRUE**/**FALSE**).

Vectors are cool. I like to think of them as boxes that can only be stacked on top of one another.

```
typeof(c(TRUE))
```

```
## [1] "logical"
```

```
typeof(c(TRUE, 1))
```

```
## [1] "double"
```

```
typeof(c(TRUE, 1, 'a'))
```

```
## [1] "character"
```

Functions

Functions are containers where anything or nothing can happen, but whatever happens, it happens the same way every single time. They allow for generalization of complicated ideas and routines that we wish to repeat over and over again. A function may have an input, but no output. It may have an output, but no input, both or none. If it's something you need to do repeatedly, or

containing code makes your program easier to read, then write a function for that process.

Rule 4: Functions have inputs, outputs, and a body. A function can have multiple outputs, but given a particular set of inputs, the solution should never change assuming you are not developing a function with randomness built in.

R has a built-in constant called `letters`. This means that no matter where you are writing R, `letters` will be available to you. We see that `letters` is a **character vector** in our program below, and use the composition of functions to create a program that describes `letters`.

```
print(letters)
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
```

Next, we can use some functions which take in pretty much any object that exists in R and spits back information regarding the `letters` data.

```
main <- function() {
  print_information <- function(x) {

    variable_name = deparse1(substitute(x))

    length_x = length(x)
    typeof_x <- typeof(x)
    is_vec_x <- is.vector(x)

    meta_list <- list(
      length = length_x,
      type = typeof_x,
      is_vector = is_vec_x
    )

    cli::cli_alert('Information about {variable_name}')

    cli::cli_alert_info("{variable_name} is a 1x{length_x} dimensional")
    cli::cli_alert_info("")

    purrr::iwalk(meta_list, function(x, index) {
      cli::cli_alert_info(glue::glue('{index} {x} is type {typeof(x)}'))
    })

    return(meta_list)
```

```
  }

  cli::cli_alert_info('Execute print_information')
  output <- print_information(mtcars)
  cli::cli_alert_success('Execute print_information complete')

  print(output)
}

main()
```

```
## i Execute print_information

## > Information about mtcars

## i mtcars is a 1x11 dimensional

## i

## i length 11 is type integer

## i type list is type character

## i is_vector FALSE is type logical

## v Execute print_information complete

## $length
## [1] 11
##
## $type
## [1] "list"
##
## $is_vector
## [1] FALSE
```

Debugging

What is the debugger?

How to learn R without knowing any R

`browser()`

`next`, `continue`

`debug` and `undebug`

`debugonce`

Understanding debugging output

LOTS OF DEBUGGING EXERCISES CANNOT STRESS ENOUGH

Vectors

`c`

`[` and `[[`

- Vectors
  - atomic
- Strings
  - Base R
  - `stringr`
    - Regular Expressions
  - Cheat Sheet
- Numbers
  - Integer
  - Double
- Factors
  - `as.factor` vs. `as_factor`
- Dates
  - Base R
  - `lubridate`

Lists

`list`

`[` and `[[`

- Lists
  - `list()` and `c`
  - `[` and `[[`
  - Connection between lists and json
    - `jsonlite`

Tables

`c`

`[` and `[[`

- Tables
  - matrices
  - `data.frame` vs `tibble`
  - data.frames are lists with equal length, atomic vectors

Functional Programming

2. Functions
    - Sequences
    - Mapping functions
    - pipes
    - void
    - `return`
        - Can a function return nothing?
        - What are side effects?
        - Multiple return statements

Base R

`apply`, `lapply`, `mapply`

Modern R

`purrr` * `map_*` * `map2_*` * `pmap_*` * Iterate over What? * Why are data.frames mapped over columnwise? * A: data.frames are lists, and mapping functions will iterate over each individual item in a list

Tidy Data

- Concept of tidy data
  - Tidy Data Paper
- `tidyr`
  - `pivot_longer`
  - `pivot_wider`

dplyr

- `dplyr` and data manipulation
  - main functions
    - `select`
    - `mutate`
    - `filter`
    - `transmute`
  - summarizing data
    - `group_by`
    - `summarize` - one row per group
    - `mutate` - one or many rows per group will have same value
    - `ungroup` - remove grouping
      - Not everything has to be a `group_by`
      - Solving group problems with vectors

- Joining Tables
  - `inner_join`
  - `full_join`
  - `left_join` / `right_join`

Project Outline

To be expanded over many chapters

1. Windows vs Mac vs Linux
2. Docker Installation
    - Windows needs to set up VM in bios
3. RStudio IDE
    - Cheat Sheet
4. reddit api creds
5. reticulate
    - Enough R to know Python
        - Type Conversions
    - miniconda installation
    - virtual environments
6. Package Structure
    - Defaults for RStudio
        - Rebuild and Restart with Roxygen2
    - `.env`
    - `.gitignore`
    - `.Rprofile`
    - `.Renviron`
    - Packages necessary for efficient development
        - usethis
        - roxygen
        - devtools
            * Cheat Sheet
    - Make and Makefiles
        - Automating Package Build
    - Unit Testing (probably bad location for ut, no code written)

    - testthat

7. Git
    - Github
    - git circle, workflow
8. Retrieving Data from API
    - `praw`
    - `dotenv` and `.env`
    - Old Reddit Code to start with
9. Docker and Docker Compose Introduction
    - `.dockerignore`
10. Create Postgres Database
    - What are Ports?
    - Postgres Credentials
11. Create functions for Storing Reddit Data
12. *Need preferred method for streaming data, i.e., Airflow not a good scheduler for scripts that are always running* and need a kickstart on failure, timeout, etc. Docker with `restart: always` may be sufficient
13. Plumber API
    - Add to docker-compose
    - Functions for ETL, Shiny Application
14. ETL with Airflow and HTTP operator connected to Plumber API
15. Shiny
    - Reactive Graph
        - Order does not matter, the graph does
    - Why Modules?
        - `map` over modules
16. Automating Infrastructure
    - `awscli`
    - `boto3`
        - `biggr`
    - Create EC2 Server from R
        - User Data