

4.6 Introduction to R: Programming

Julia Gallucci

Data Science Institute, University of Toronto

2023-12-16

Acknowledgements

Slides are adapted from Anjali Silva, originally from Amy Farrow under the supervision of Rohan Alexander, University of Toronto. Slides have been modified by Julia Gallucci, 2023.

Overview

- ▶ Functions (Wickham and Grolemund, 2017, Chapter 19)
- ▶ Loops (Wickham and Grolemund, 2017, Chapter 21)
- ▶ if/else logic (Alexander (eds), 2021, Chapter 47)
- ▶ purr
- ▶ Simulation (Alexander (eds), 2021, Chapter 47)

What you need

Packages:

- ▶ `library(ggplot2)`
- ▶ `library(purrr)`
- ▶ `library(tidyverse)`

Data:

- ▶ `iris`

Functions

Functions

Introduction

You can write your own functions in R, and you should consider doing so when you have copy-pasted a chunk of code twice.

Structure

You provide a name, inputs (also known as arguments), and the body of the function that performs the operation.

```
function_name <- function(inputs) {  
  <calculations using inputs>  
  return(outputs)  
}
```

When naming, try not to use names that already have meaning in R.

Loops

Basic form

Loops are another tool for reducing the need to duplicate code, this time by repeatedly performing a task.

1. For loops iterate over a set amount:

```
for (sequence to iterate over) {  
    <code to execute>  
}
```

2. While loops iterate based on a stopping condition:

```
while (iterator condition) {  
    <code to execute>  
}
```


For loop example

```
for (i in 1:10){  
  print(i*5)  
}
```

```
[1] 5  
[1] 10  
[1] 15  
[1] 20  
[1] 25  
[1] 30  
[1] 35  
[1] 40  
[1] 45  
[1] 50
```

For loops to modify an existing object

To create new column that adds the Sepal.Length in each row with the Sepal.Length from the previous row:

```
for (i in 2:nrow(iris)) {  
  iris$previous_combo[i] <-  
  iris$Sepal.Length[i] +  
  iris$Sepal.Length[i-1]  
}  
iris
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	previous_combo
1	5.1	3.5	1.4	0.2	setosa	NA
2	4.9	3.0	1.4	0.2	setosa	10.0
3	4.7	3.2	1.3	0.2	setosa	9.6
4	4.6	3.1	1.5	0.2	setosa	9.3
5	5.0	3.6	1.4	0.2	setosa	9.6
6	5.4	3.9	1.7	0.4	setosa	10.4
7	4.6	3.4	1.4	0.3	setosa	10.0
8	5.0	3.4	1.5	0.2	setosa	9.6
9	4.4	2.9	1.4	0.2	setosa	9.4

Different ways to loop:

You can loop over elements:

```
for (i in c("a", "b", "c")){  
  print(i)  
}
```

```
[1] "a"
```

```
[1] "b"
```

```
[1] "c"
```

You can loop over numeric indices:

```
for (i in 1:3) {  
  print(now( ) + i)  
}
```

```
[1] "2023-06-08 03:08:07 UTC"
```

```
[1] "2023-06-08 03:08:08 UTC"
```

```
[1] "2023-06-08 03:08:09 UTC"
```

Using a vector to collect outputs

```
outputs <- c()

for (i in 1:5) {
  outputs <- c(outputs, i) * i
}

outputs
```

```
[1] 120 240 180 80 25
```

While loop example

Note that we initiate the iterator `i` outside the loop and increment it in the loop. If the iterator never increases in the loop, then the loop will never end.

```
i = 1
while(i <= 10){
    print(i*5)
    i = i + 1
}
```

```
[1] 5
[1] 10
[1] 15
[1] 20
[1] 25
[1] 30
[1] 35
[1] 40
[1] 45
[1] 50
```

If/else Logic

Basic structure

```
if(condition1) {  
  <code to execute if condition1 is TRUE>  
} elif (condition2) {  
  <code to execute if condition1 is FALSE  
  and conditions2 is TRUE>  
} else {  
  <code to execute if condition1 and  
  condition2 are both FALSE>  
}
```

Conditions

Conditions must either evaluate to TRUE or FALSE.

You can combine multiple conditions using the 'or' operator:

▶ `(condition1) || (condition2)`

You can combine multiple conditions using the 'and' operator:

▶ `(condition1) && (condition2)`

To find out if any of a list of conditions is TRUE, use `any()`.

To find out if all of a list of conditions is TRUE, use `all()`.

if else function

The function `if_else` writes out a conditional statement in one line.

```
if_else(condition, output if condition  
is TRUE, output if condition is FALSE)
```


Case when

When you have a list of possible conditions, you can use `case_when` instead.

```
case_when(condition1 ~ output1,  
          condition2 ~ output2,  
          condition3 ~ output3,  
          ...  
          )
```

Example

```
grades <- tibble(grade = c(94, 87, 73))
```

```
grades %>%  
  mutate(letter = case_when(  
    grade >= 80 ~ "A",  
    grade >= 70 ~ "B",  
    TRUE ~ "F"))
```

```
# A tibble: 3 x 2
```

```
  grade letter
```

```
  <dbl> <chr>
```

```
1     94 A
```

```
2     87 A
```

```
3     73 B
```

Note that each condition is checked in order: if condition1 is TRUE, output1 will be chosen and condition2 will not be checked.

purrr

Iteration is made more straightforward with the purrr library.

Mapping functions

Each type of output has a different function:

- ▶ `map()` for lists
- ▶ `map_lgl()` for logical vectors
- ▶ `map_int()` for integer vectors
- ▶ `map_dbl()` for double vectors
- ▶ `map_chr()` for character vectors

Looping over columns in a dataset

```
iris %>%  
  map_dbl(mean)
```

```
Sepal.Length Sepal.Width Petal.Length  
Petal.Width Species  
5.843333 3.057333 3.758000 1.199333 NA  
previous_combo  
      NA
```

```
iris %>%  
  map_chr(typeof)
```

```
Sepal.Length Sepal.Width Petal.Length  
Petal.Width Species  
"double" "double" "double" "double"  
"integer"  
previous_combo  
      "double"
```

Looping over columns in a dataset

```
iris %>%  
  map(summary)
```

\$Sepal.Length

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
4.300	5.100	5.800	5.843	6.400	7.900

\$Sepal.Width

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
2.000	2.800	3.000	3.057	3.300	4.400

\$Petal.Length

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1.000	1.600	4.350	3.758	5.100	6.900

\$Petal.Width

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.100	0.300	1.300	1.199	1.800	2.500

\$Species

setosa versicolor virginica

Mapping over multiple arguments

```
x <- list(1, 1, 1)
y <- list(10, 20, 30)

map2(x, y, ~ .x + .y)
```

```
[[1]]
[1] 11
```

```
[[2]]
[1] 21
```

```
[[3]]
[1] 31
```

Questions?

Simulation

Simulation

We can generate random data in R.

```
runif(5)
```

```
[1] 0.8848765 0.2726858 0.9015277  
0.7174576 0.3329381
```

```
runif(5)
```

```
[1] 0.06293288 0.13307987 0.68781301  
0.30197794 0.70075858
```

The outcomes will be different every time.

Simulation

If you want the results to be consistent, you must set a seed. The seed can be any number.

```
set.seed(1818)  
runif(5)
```

```
[1] 0.1763119 0.9955676 0.5480822  
0.7362859 0.6225994
```

```
set.seed(1838)  
runif(5)
```

```
[1] 0.07697791 0.06472722 0.41493940  
0.85446386 0.24067640
```

```
set.seed(1818)  
runif(5)
```

```
[1] 0.1763119 0.9955676 0.5480822  
0.7362859 0.6225994
```

The uniform distribution

```
runif(number, min, max)
```

```
set.seed(1818)
```

```
runif(10, 1, 20)
```

```
[1] 4.349927 19.915784 11.413561  
14.989433 12.829389 15.445609 7.815725  
[8] 11.646421 8.964373 19.284247
```

The normal distribution

```
rnorm(number, mean, sd)
```

```
set.seed(1818)  
rnorm(10, 5, 1)
```

```
[1] 4.070488 5.120817 5.312315 4.638124  
4.796002 5.437974 3.674402 5.231550  
[9] 5.093735 6.607725
```

Sampling

```
sample(thing to sample from, size =  
number, replace, prob = vector of  
probability weights)
```

```
set.seed(1818)  
sample(c("a", "b", "c"),  
       size = 10,  
       replace = TRUE,  
       prob = c(0.1, 0.2, 0.7))
```

```
[1] "c" "a" "c" "b" "c" "b" "c" "c" "c"  
"a"
```

The probability weights are optional. If you do not specify, all the results will be equally probable.

If you specify `replace = FALSE`, there must be as many or more in the thing that you sample from as the desired sample size.

Simulating datasets

We can put our randomization skills to use and create toy datasets.

```
set.seed(1818)
simulated_data <- tibble(X = runif(10,
0, 20),
  Y = 3*X + rnorm(10, 0, 1))
```

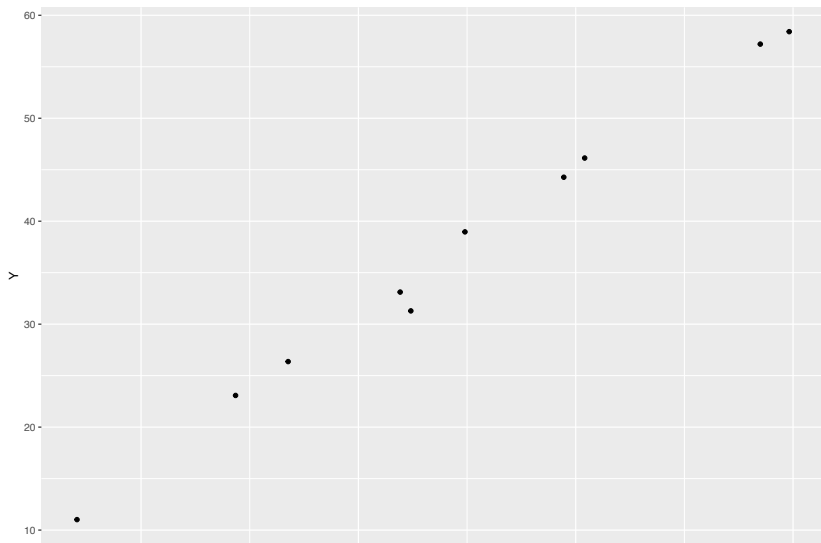
```
simulated_data
```

```
# A tibble: 10 x 2
```

	X	Y
	<dbl>	<dbl>
1	3.53	11.0
2	19.9	58.4
3	11.0	33.1
4	14.7	44.3
5	12.5	39.0
6	15.2	46.1
7	7.17	23.1
8	11.2	31.3
9	8.38	26.4

Simulating datasets

```
simulated_data %>%  
  ggplot(aes(x = X, y = Y)) +  
  geom_point()
```



Exercises

Exercises

1-Write a greeting function that says “good morning”, “good afternoon”, or “good evening”, depending on the time of day.

2-Simulate a dataset using a normal distribution with mean 100 and standard deviation 15 as variable X , and a quadratic transformation of X as variable Y . Graph your data.

Questions?