Iterations: For Loops

R Programming Structures

Gaston Sanchez

R Coding Compendium

Donate

Introduction

Before describing some of the common programming structures in R, we need to talk about a basic concept called **Expressions**.

You've been using simple expressions so far, but we need to introduce the notion of a compound expression.

Loops

Loop

R code is composed of a series of **expressions**

- Many times we need to perform a procedure **several times**
- The main idea is that of **iteration**
- For this purpose we use loops
- We perform operations as long as some condition is fulfilled
- R provides three basic paradigms:
  - `for`
  - `repeat`
  - `while`

Big Favor

In order to learn about loops, I'm going to ask you to **forget about vectorization**.

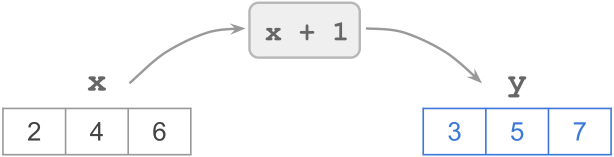
Instead, let's describe how to perform those type of operations "manually", step by step.

Vectorization Reminder

A vectorized computation is any computation that when applied to a vector operates on all of its elements

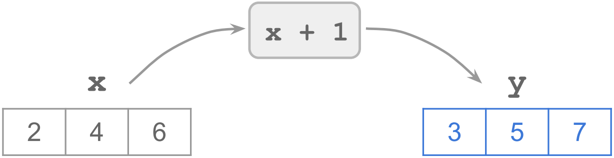
```r
# examples of vectorized code
c(1, 2, 3) + c(3, 2, 1)
c(1, 2, 3) * 3
abs(c(-1, 2, 0))
```

What if R did not have vectorized code?



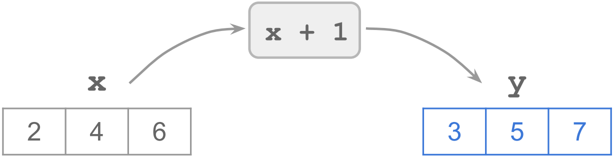
*How to get **x+1**, step-by-step, without using vectorization?*

What if R did not have vectorized code?



```
# initialize empty y
y <- rep(0, 3)

y[1] <- x[1] + 1
y[2] <- x[2] + 1
y[3] <- x[3] + 1
```

What if R did not have vectorized code?

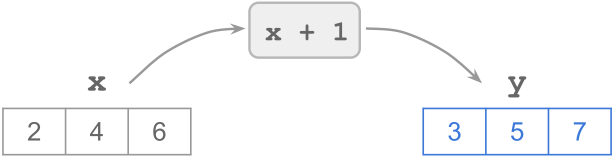


```
# initialize empty y
y <- rep(0, 3)
```

```
y[1] <- x[1] + 1
y[2] <- x[2] + 1
y[3] <- x[3] + 1
```

What do all of these commands have in common?

What if R did not have vectorized code?



```
# initialize empty y
y <- rep(0, 3)

y[1] <- x[1] + 1
y[2] <- x[2] + 1
y[3] <- x[3] + 1
```

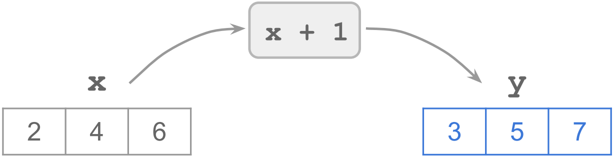
They all have the same format:

```
y[pos] <- x[pos] + 1
```

(`pos` indicates position)

Let's use a **for** loop

Using a for loop



x

2	4	6

x + 1

y

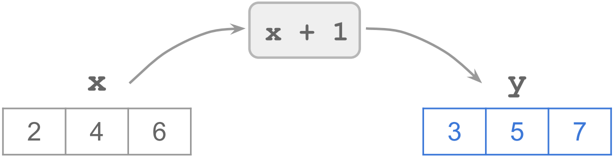
3	5	7

```
# initialize empty y
y <- rep(0, 3)
```

Step 1

```
  y[pos] <- x[pos] + 1
```

Using a for loop

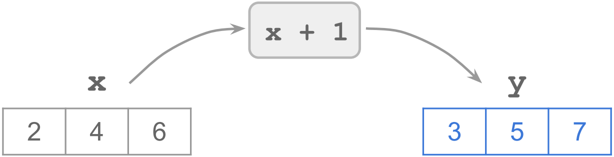


```
# initialize empty y
y <- rep(0, 3)

for (              ) {
  y[pos] <- x[pos] + 1
}
```

Step 1

Step 2

Using a for loop



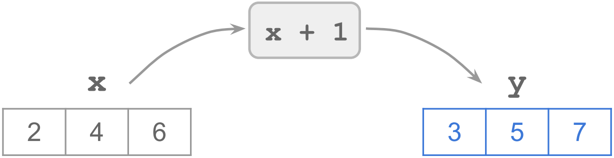
```
# initialize empty y
y <- rep(0, 3)

for (pos in      ) {
  y[pos] <- x[pos] + 1
}
```

Step 1

Step 2

Step 3

Using a for loop


x + 1

x

2	4	6

y

3	5	7

```r
# initialize empty y
y <- rep(0, 3)

for (pos in 1:3) {
  y[pos] <- x[pos] + 1
}
```

Step 1

Step 2

Step 3

Step 4

Anatomy of a for loop

Anatomy of a for loop

```
x <- c(2, 4, 6)
y <- rep(0, 3)

for (pos in 1:3) {
  y[pos] <- x[pos] + 1
}
```

Anatomy of a for loop

```
x <- c(2, 4, 6)
y <- rep(0, 3)
```

for statement

```
for (pos in 1:3) {
  y[pos] <- x[pos] + 1
}
```

Anatomy of a for loop

```
x <- c(2, 4, 6)
y <- rep(0, 3)
```

Iterator: auxiliary variable

```
for (pos in 1:3) {
  y[pos] <- x[pos] + 1
}
```

Anatomy of a for loop

```
x <- c(2, 4, 6)
y <- rep(0, 3)
```

Iterator: auxiliary variable

```
for (pos in 1:3) {
  y[pos] <- x[pos] + 1
}
```

Anatomy of a for loop

```
x <- c(2, 4, 6)
y <- rep(0, 3)
```

“in” keyword

```
for (pos in 1:3) {
  y[pos] <- x[pos] + 1
}
```

Anatomy of a for loop

```
x <- c(2, 4, 6)
y <- rep(0, 3)
```

Vector of “times”

```
for (pos in 1:3) {
  y[pos] <- x[pos] + 1
}
```

Anatomy of a for loop

```r
x <- c(2, 4, 6)
y <- rep(0, 3)

for (pos in 1:3) {
  y[pos] <- x[pos] + 1
}
```

Anatomy of a `for()` loop

- You use the `for()` function
- Inside parenthesis, you need three ingredients:
  - an auxiliary iterator, e.g. `s`
  - the keyword `in`
  - a vector to iterate through, e.g. `1:10`
- The code for the repetitive steps gets wrapped inside braces
- R will automatically handle the auxiliary iterator (no need to explicitly increase its value)
- The length of the iterations vector determines the number of times the code inside the loop has to be repeated
- You use **for** loops when you know how many times a series of calculations need to be repeated.

In Summary ...

For Loops

Often we want to repeatedly carry out some computation a fixed number of times. For instance, repeat an operation for each element of a vector. In R this is done with a **for** loop

About For Loops

for loops are used when we know exactly how many times we want the code to repeat

```
for (iterator in times) {
  do_something
}
```

for takes an *iterator* variable and a vector of *times* to iterate through

For Loops

The vector of *times* does not have to be a numeric vector; it can be any vector

```r
value <- 2
times <- c('a', 'b', 'c', 'd', 'e')

for (i in times) {
  value <- value * 2
  print(value)
}
```

```
## [1] 4
## [1] 8
## [1] 16
## [1] 32
## [1] 64
```

For Loops and Next statement

Sometimes we need to skip a loop iteration if a given condition is met, this can be done with a `next` statement

```r
for (iterator in times) {
  expr1
  expr2
  if (condition) {
    next
  }
  expr3
  expr4
}
```

For Loops and Next statement

```r
x <- 2
for (i in 1:5) {
  y <- x * i
  if (y == 8) {
    next
  }
  print(y)
}
```

```
## [1] 2
## [1] 4
## [1] 6
## [1] 10
```

Nested Loops

It is common to have nested loops

```
for (iterator1 in times1) {
  for (iterator2 in times2) {
    expr1
    expr2
    ...
  }
}
```

Nested Loops: example

```r
A <- matrix(1:12, nrow = 3, ncol = 4)

# obtaining reciprocal of those values < 6
for (i in 1:nrow(A)) {
  for (j in 1:ncol(A)) {
    if (A[i,j] < 6) A[i,j] <- 1 / A[i,j]
  }
}

A
```

```
##           [,1] [,2] [,3] [,4]
## [1,] 1.0000000 0.25    7   10
## [2,] 0.5000000 0.20    8   11
## [3,] 0.3333333 6.00    9   12
```

More about for loops

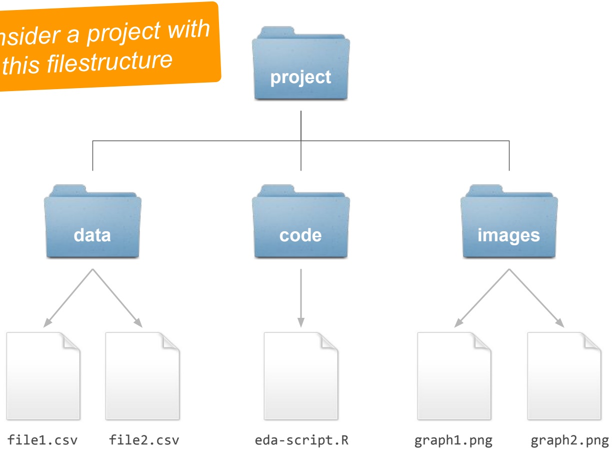
Example

In this part I want to discuss a couple of examples that involve using loops.

The idea is to go through less basic (and more interesting) cases for working with loops.

Motivation

Consider a project with this filestructure



Filestructure

```
project/
        data/
            file1.csv
            file2.csv
        code/
            eda-script.R
        images/
            graph1.png
            graph2.png
```

Hypothetical Project

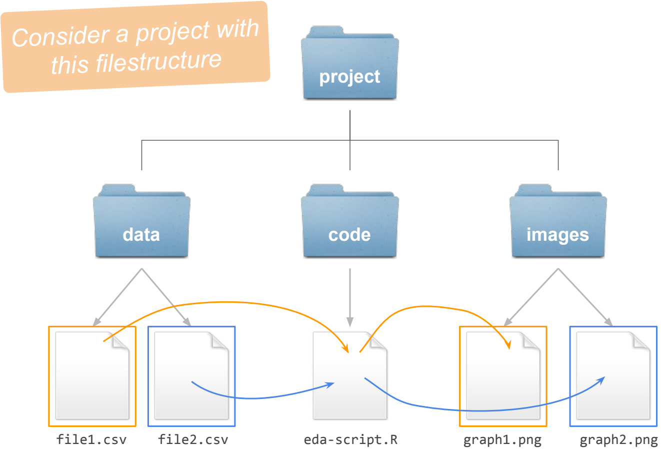
Say you have a couple of CSV data files, which are supposed to be your “raw” data files.

As part of the Data Preparation stage, you will have to do a little bit of exploratory data analysis (eda), e.g. creating scatterplots for each data file.

Also, suppose you will use the eda-script.R file to write the code for EDA.

This obviously involves importing (reading in) the CSV files, and making the graphs.

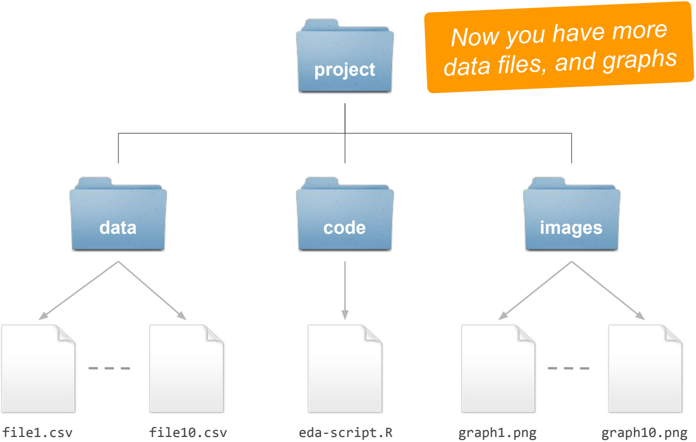
For loop example



```r
# importing raw data files
raw1 <- read.csv("../data/file1.csv")
raw2 <- read.csv("../data/file2.csv")

# scatterplots
graph1 <- ggplot(data = raw1) +
geom_point(aes(x = A, y = B)) +
labs(title = "scatter 1")
ggsave("../images/graph1.png", graph1)

graph2 <- ggplot(data = raw2) +
geom_point(aes(x = A, y = B)) +
labs(title = "scatter 2")
ggsave("../images/graph2.png", graph2)
```

What if you had to do the same tasks for 5, or 10, or 100 data files?

For loop example



Now you have more data files, and graphs

Too Much Repetition

```r
# importing raw data files
raw1 <- read.csv("../data/file1.csv")
raw2 <- read.csv("../data/file2.csv")
raw3 <- read.csv("../data/file3.csv")
raw4 <- read.csv("../data/file4.csv")
raw5 <- read.csv("../data/file5.csv")
raw6 <- read.csv("../data/file6.csv")
raw7 <- read.csv("../data/file7.csv")
raw8 <- read.csv("../data/file8.csv")
raw9 <- read.csv("../data/file9.csv")
raw10 <- read.csv("../data/file10.csv")
```

Imagine if you had 100 (or more) files. This is labor intensive, time consuming, error prone, boring . . . DON'T do this!

Let's use a for loop

Too Much Repetition

```r
# importing raw data files
raw1 <- read.csv("../data/file1.csv")
raw2 <- read.csv("../data/file2.csv")
raw3 <- read.csv("../data/file3.csv")
raw4 <- read.csv("../data/file4.csv")
raw5 <- read.csv("../data/file5.csv")
# ... etc
```

What do all these commands have in common?

Creating file names (by-hand)

```r
# creating file names
paste0("../data/file", 1, ".csv")

paste0("../data/file", 2, ".csv")

paste0("../data/file", 3, ".csv")

...

paste0("../data/file", 10, ".csv")
```

For loop example

```
# creating file names
paste0("../data/file", 1, ".csv")

paste0("../data/file", 2, ".csv")

paste0("../data/file", 3, ".csv")

...

paste0("../data/file", 10, ".csv")
```

For loop example

eda-script.R

```
for (num in 1:10) {
  # importing file
  filepath <- paste0("../data/file",
                     num,
                     ".csv")
  dat <- read.csv(filepath)

  # scatterplot
  ...
}
```

Code in next slide

For loop example

eda-script.R

```
for (num in 1:10) {
  # importing file
  ...

  # scatterplot
  scat <- paste0("scatter ", num)
  graph <- ggplot(data = dat) +
    geom_point(aes(x = A, y = B)) +
    labs(title = scat)




}
```

Code in previous slide

For loop example

eda-script.R

```
for (num in 1:10) {
  # importing file
  ...

  # scatterplot
  scat <- paste0("scatter ", num)
  graph <- ggplot(data = dat) +
    geom_point(aes(x = A, y = B)) +
    labs(title = scat)

  gfile <- paste0("../images/graph",
                  num, ".png")
  ggsave(gfile, graph)
}
```

Code in previous slide

For loop example

Putting it all together

eda-script.R

```
for (num in 1:10) {
   # importing file
   filepath <- paste0("../data/file", num, ".csv")
   dat <- read.csv(filepath)

   # scatterplot
   scat <- paste0("scatter ", num)
   graph <- ggplot(data = dat) +
      geom_point(aes(x = A, y = B)) +
      labs(title = scat)

   gfile <- paste0("../images/graph", num, ".png")
   ggsave(gfile, graph)
}
```