CODING LOOPS - "break" - "next" - "repeat"

DSC 205 - Advanced introduction to data science

Marcus Birkenkrahe

February 24, 2023

README

Figure 1: Photo by Frank Leuderalbert on Unsplash

- Download the **raw** file to practice during the lecture from GitHub, save it as `8_loop_break_practice.org` and upload it to Canvas later.
- To test your Emacs mettle, open it on the CMD line with the command `emacs -nw` (no graphics - not needed for this exercise).

Declaring `break` or `next`

- `for` loops will exit only when the *loopindex* exhausts the *loopvector*
- `while` loops will exit only when the *loopcondition* evaluates to `FALSE`
- `break` allows to pre-emptively terminate a loop
- `next` allows to leave a loop and continue execution
- Both `break` and `next` work the same way in `for` or `while` loops

Example: `break`

- Divide a number `foo` by each element in a `numeric` vector `bar`:

```
foo <- 5
bar <- c(2,3,1.1,4,0,4.1,3)
```

- You want to halt execution if one of the results evaluates to `Inf`:
  1. check each iteration with `is.finite`
  2. if a check evaluates to `TRUE`, terminate loop with `break`

```
foo <- 5
bar <- c(2,3,1.1,4,0,4.1,3)
## initialize results
loop1.result <- rep(NA,length(bar))
loop1.result
bar
## loop over length of bar
for (i in 1:length(bar)) {
  temp <- foo/bar[i]
  if (is.finite(temp)) {
    loop1.result[i] <- temp
  } else {
    break
  } # end if
} # end for
loop1.result
```

```
[1] NA NA NA NA NA NA NA
[1] 2.0 3.0 1.1 4.0 0.0 4.1 3.0
[1] 2.500000 1.666667 4.545455 1.250000       NA       NA       NA

foo <- 5
bar <- c(2,3,1.1,4,0,4.1,3)
## initialize results
loop1.init <- rep(NA,length(bar))
loop1.result <- loop1.init
loop1.result
## loop over length of bar
for(i in 1:length(bar)) {
  loop1.result[i] <- foo/bar[i]
}
loop1.result
loop1.result <- loop1.init
## with break and condition
for (i in 1:length(bar)) {
  temp <- foo/bar[i]
  if (is.finite(temp)) {
    loop1.result[i] <- foo/bar[i]
  } else {
    break
  }
}
loop1.result

[1] NA NA NA NA NA NA NA
[1] 2.500000 1.666667 4.545455 1.250000      Inf 1.219512 1.666667
[1] 2.500000 1.666667 4.545455 1.250000       NA       NA       NA
```

Example: `next`

- For more routine operations, use `next` instead, which simply advances to the next iteration and continues execution
- Here, `next` avoids division by zero:

```
foo <- 5
bar <- c(2,3,1.1,4,0,4.1,3)
```

```
## initialize results
loop2.result <- rep(NA,length(bar))
loop2.result
## loop over length of bar
for (i in 1:length(bar)) {
  if (bar[i]==0) {
    next
  } # end if
  loop2.result[i] <- foo/bar[i]
} # end for
loop2.result
```

```
[1] NA NA NA NA NA NA NA
[1] 2.500000 1.666667 4.545455 1.250000       NA 1.219512 1.666667
```

`break` and `next` in nested loops

- If you use either `break` or `next` in a nested loop, the command will apply only to the innermost loop.
- Fill a matrix with multiples of two vectors and use `next` in the inner loop to skip certain values:

```
loopvec1 <- 5:7
loopvec2 <- 9:6
baz <- matrix(NA,
              length(loopvec1),
              length(loopvec2))
```

- Loop over both vectors, exclude loops where their element-wise product is greater or equal than 54:

```
loopvec1 <- 5:7
loopvec2 <- 9:6
baz <- matrix(NA,
              length(loopvec1),
              length(loopvec2))
for (i in 1:length(loopvec1)) {
  for (j in 1:length(loopvec2)) {
    temp <- loopvec1[i] * loopvec2[j]
```

```
    if (temp >= 54) {
      next # leave inner loop
    } #end if
    baz[i,j] <- temp
  } # end for i
} # end for j
baz
```

```
     [,1] [,2] [,3] [,4]
[1,]   45   40   35   30
[2,]   NA   48   42   36
[3,]   NA   NA   49   42
```

Repeating operations with `repeat`

- The template for `repeat` is simple - it repeats whatever stands between the curly braces:

```
repeat {
  do any code in here
}
```

```
Error: unexpected symbol in:
"repeat {
  do any"
Error: unexpected '}' in "}"
```

- Repetition with `repeat` does not include a *loopindex* or *loopcondition*. To stop repeating the code, you need `break`.

Example: `repeat`

- The Fibonacci series is an infinite series of integers beginning with 1,1,2,3,5,8,13,...formally: the n-th Fibonacci number F_{nx} is $F_n = F_{n-2} + F_{n-1}$, $n = 2, 3, 4, 5$ and $F_1 = F_2 = 1$.
- You can use `repeat`, and `break` out of the loop:

```
fib.a <- 1  # initialize first two terms
fib.b <- 1
repeat {
  temp <- fib.a + fib.b   # compute next term
  fib.a <- fib.b          # move variables forward
  fib.b <- temp           # fib.b becomes new Fibonacci number
  cat(fib.b,",",sep="")   # print Fibonacci number
  if (fib.b > 150) {      # cut of if number greater than 150
    cat("Break now...\n")
    break                 # leave repeat loop
  } # end if
} #end repeat

2,3,5,8,13,21,34,55,89,144,233,Break now...
```

TODO Exercises

Submit solutions to these exercises as Org-mode files for bonus.

`while` without `break` or `next`

Earlier, we divided `foo` by `bar`, where:

```
foo <- 5
bar <- c(2,3,1.1,4,0,4.1,3)
foo
bar
```

```
[1] 5
[1] 2.0 3.0 1.1 4.0 0.0 4.1 3.0
```

1. Write a `while` loop - without using `break` or `next` that will produce the same vector as `loop1.result` (see GitHub): compute `foo/bar` and make sure you break off as soon as `Inf` is produced.

   ```
   foo <- 5
   bar <- c(2,3,1.1,4,0,4.1,3)
   ## initialize results
   loop1.result <- rep(NA,length(bar))
   loop1.result
   bar
   ## loop over length of bar
   for (i in 1:length(bar)) {
     temp <- foo/bar[i]
     if (is.finite(temp)) {
       loop1.result[i] <- temp
     } else {
       break
     } # end if
   } # end for
   loop1.result
   ```

   ```
   [1] NA NA NA NA NA NA NA
   [1] 2.0 3.0 1.1 4.0 0.0 4.1 3.0
   [1] 2.500000 1.666667 4.545455 1.250000       NA       NA       NA
   ```

2. Obtain the same result as `loop2.result` using an `ifelse` function instead of a loop.

```
foo <- 5
bar <- c(2,3,1.1,4,0,4.1,3)
## initialize results
loop2.result <- rep(NA,length(bar))
loop2.result
## loop over length of bar
for (i in 1:length(bar)) {
  if (bar[i]==0) {
    next
  } # end if
  loop2.result[i] <- foo/bar[i]
} # end for
loop2.result
```

```
[1] NA NA NA NA NA NA NA
[1] 2.500000 1.666667 4.545455 1.250000       NA 1.219512 1.666667
```

`for` and `repeat` instead of `while`

To demonstrate `while` loops, you used `mynumbers` to progressively fill `mylist` with identity matrices whose dimensions matched the values in `mynumbers`. The loop was instructed to stop when it reached the end of the `numeric` vector or a number greater than 5:

```
mylist <- list()  # create an empty list to store all matrices
counter <- 1      # set loop index counter variable to 1
mynumbers <- c(4,5,1,2,6,2,4,6,6,2) # matrix dimensions
mycondition <- mynumbers[counter] <= 5 # while loop condition

while (mycondition) {
  mylist[[counter]] <- diag(mynumbers[counter]) # add matrix to list
  counter <- counter + 1   # increase counter (stepping through mynumbers)
  ## update loop condition
  if (counter <= length(mynumbers)) {
    mycondition <- mynumbers[counter] <= 5  # counter in bounds
  } else {
    mycondition <- FALSE   # counter out of bounds (end of mynumbers)
  }
}
mylist
```

```
[[1]]
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    1    0    0
[3,]    0    0    1    0
[4,]    0    0    0    1

[[2]]
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    0    0    0
[2,]    0    1    0    0    0
[3,]    0    0    1    0    0
[4,]    0    0    0    1    0
[5,]    0    0    0    0    1

[[3]]
     [,1]
[1,]    1

[[4]]
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

1. Write a `for` loop using a `break` declaration that does the same thing.
2. Write a `repeat` statement that does the same thing.

Glossary

TERM	MEANING
`break`	leave loop and stop execution
`next`	leave current loop and continue execution
`repeat`	repeat any statements in the loop area

References

- Ceballos, M. (2013). Data structure. URL: venus.ifca.unican.es.
- Davies, T.D. (2016). The Book of R. NoStarch Press.

- Treadway, A. (20 Oct 2020). Why you should use vapply in R. URL: theautomatic.net.
- Zach (Dec 7, 2021). How to Use the mapply() Function in R (With Examples). URL: statology.org.