CODING LOOPS WITH "while" - LECTURE

DSC 205 - Advanced introduction to data science

Marcus Birkenkrahe February 15, 2023

README



Figure 1: Photo by La-Rel Easter on Unsplash

Download the raw practice file from GitHub and save it as 6_loop_while_practice.org. To test your Emacs mettle, open it on the CMD line with the command emacs -nw (no graphics - not needed for this exercise).

while loops

• A while loop repeats code until a condition evaluates as FALSE:

```
while (loopcondition) {
  do any code in here
}
```

- This means to avoid *infinite loops*, the operations in the braced area must cause the loop to exit.
- Exiting a loop works if either the *loopcondition*/ is FALSE, or if a break command is met.
- To escape infinite loops in Emacs, enter C-g in the Rterm or Rgui console (outside Emacs) enter C-c or <ESC>.

Simple example

- 1. Set the condition variable myval to 5
- 2. Test if myval is less than 10
- 3. If it is, increase myval by 1
- 4. Print the current value of myval using cat on one line
- 5. Print the current value of the condition with cat on the next line

```
myval <- 5
while(myval<10) {
  myval <- myval + 1
  cat("\nmyval is now",myval,"\n")
  cat("condition is now",myval<10,"\n")
}

myval is now 6
condition is now TRUE

myval is now 7
condition is now TRUE</pre>
```

```
myval is now 8
condition is now TRUE
myval is now 9
condition is now TRUE
myval is now 10
condition is now FALSE
```

Extended example

- It is often useful to set the *loopcondition* to be an object so that you can modify it inside the braced area.
- In the example, you will use a while loop to iterate over an integer vector mynumbers and create an *identity matrix* using diag with the dimension dim matching the current integer.
- This loop should stop when it reaches a number in the vector mynumbers that's greater than 5, or when it reaches the end of the vector. The while condition is stored in a separate object mycondition.
- Create a few initial objects first:

```
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
```

• The diag function extracts or replaces the diagonal of a matrix, or constructs a diagonal matrix. Check out its arguments:

```
args(diag)
function (x = 1, nrow, ncol, names = TRUE)
NULL
```

• To test the function, create a 3x2 matrix of 0 values m and then use diag to turn it into an *identity* matrix.

```
m <- matrix(0,3,3)</pre>
diag(m) <- 1
      [,1] [,2] [,3]
[1,]
[2,]
         0
[3,]
               0
         0
     [,1] [,2] [,3]
[1,]
        1
[2,]
         0
                     0
               1
[3,]
               0
                     1
```

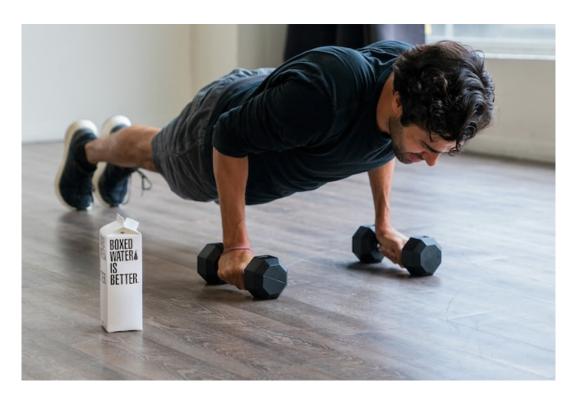
• Create an empty list 1 and add a 2x2 identity matrix to it by overwriting the first element of 1 with diag(2)

• Create the loop:

```
mycondition <- FALSE # counter out of bounds
  }
}
mylist
[[1]]
     [,1] [,2] [,3] [,4]
[1,]
              0
[2,]
                         0
        0
              1
                   0
[3,]
        0
              0
                   1
                         0
[4,]
        0
              0
                   0
                         1
[[2]]
     [,1] [,2] [,3] [,4] [,5]
[1,]
                         0
[2,]
                         0
                              0
        0
              1
                   0
[3,]
        0
              0
                         0
                              0
[4,]
        0
              0
                   0
                         1
                              0
[5,]
        0
              0
                   0
                         0
                              1
[[3]]
     [,1]
[1,] 1
[[4]]
     [,1] [,2]
[1,]
        1
[2,]
              1
        0
```

• The result is a list mylist with four members because 4 is the last element of mynumbers not greater than 5. The identity matrices have dimension 4 x 4, 5 x 5, 1 x 1 and 2 x 2.

TODO Exercises



Download the \mathbf{raw} exercise file from GitHub and save it as $6_loop_while_exercise.org$. When done, upload the file to Canvas.

TODO Glossary

TERM MEANING

References

• Davies, T.D. (2016). The Book of R. NoStarch Press.