Data Science 2

Introduction to advanced data science - spring 2023

February 20, 2023

5_loop_for_exercise.org

Make a `for` loop more efficient

Rewrite the nested loop example from the lecture where the matrix `foo` was filled with the multiples of `loopvec1` and `loopvec2`, using only a **single `for`** loop.

- Sample data:

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

```
[1] 5 6 7
[1] 9 8 7 6
     [,1] [,2] [,3] [,4]
[1,]   NA   NA   NA   NA
[2,]   NA   NA   NA   NA
[3,]   NA   NA   NA   NA
```

- Nested loop example (two `for` loops) and output

```
for (i in 1:length(loopvec1)) {
   for (j in 1:length(loopvec2)) {
    foo[i,j] <- loopvec1[i] * loopvec2[j]
  }
}

#+RESULTS:
:      [,1] [,2] [,3] [,4]
: [1,]   45   40   35   30
: [2,]   54   48   42   36
: [3,]   63   56   49   42
```

- Solution:

```
loopvec1 <- 5:7; loopvec1
loopvec2 <- 9:6; loopvec2
foo <- matrix(NA,
              length(loopvec1),
              length(loopvec2))
foo   ## reinitialize matrix foo
for (i in 1:length(loopvec1)) {
  foo[i,] <- loopvec1[i] * loopvec2
}
foo

[1] 5 6 7
[1] 9 8 7 6
     [,1] [,2] [,3] [,4]
[1,]   NA   NA   NA   NA
[2,]   NA   NA   NA   NA
[3,]   NA   NA   NA   NA
     [,1] [,2] [,3] [,4]
[1,]   45   40   35   30
[2,]   54   48   42   36
[3,]   63   56   49   42
```

- Works the other way around as well (column-wise):

```
loopvec1 <- 5:7; loopvec1
loopvec2 <- 9:6; loopvec2
foo <- matrix(NA,
              length(loopvec1),
              length(loopvec2))
foo   ## reinitialize matrix foo
for (j in 1:length(loopvec2)) {
  foo[,j] <- loopvec1 * loopvec2[j]
}
foo

[1] 5 6 7
[1] 9 8 7 6
     [,1] [,2] [,3] [,4]
[1,]   NA   NA   NA   NA
[2,]   NA   NA   NA   NA
[3,]   NA   NA   NA   NA
     [,1] [,2] [,3] [,4]
[1,]   45   40   35   30
[2,]   54   48   42   36
[3,]   63   56   49   42
```

`for` loop with `switch`

- To return a number based on the supplied value of a single `character` string, you can use the `switch` command - but it won't work if the `EXPR` is a `character` vector!

```
mystring = "Lisa"
switch(
  EXPR = mystring,
  Homer=12,
  Marge=34,
  Bart=56,
  Lisa=78,
  Maggie=90,
  NA)

[1] 78
```

- Write some code that will take a `character` vector `mystrings` and return a vector `mynums` of the appropriate `numeric` values. Then test it on this vector:

```
mystrings <- c("Peter", "Homer", "Lois", "Stewie", "Maggie", "Bart")
```

The output of your code should look like this:

- *Tip:* (1) initialize `character` vector `mystrings`, (2) initialize `numeric` vector `mynums` with missing values, (3) loop and overwrite `mynums` with the numbers corresponding to the names using `switch` for each value of `mystring`.

- Solution:

```
## initialize character vector
mystrings <- c("Peter", "Homer", "Lois", "Stewie", "Maggie", "Bart")
## initialize numeric vector
mynums <- rep(NA, length(mystrings))
## loop over mystrings and overwrite mynums
for (i in 1:length(mystrings)) {
  mynums[i] <- switch(EXPR=mystrings[i],
                      Homer=12,
                      Marge=34,
                      Bart=56,
                      Lisa=78,
                      Maggie=90,
                      NA)
}
## print mynums
mynums
```

```
[1] NA 12 NA NA 90 56
```

- If you only want to run the `switch` function, you need to put a `print` statement around it to get a result:

```
for (i in 1:length(mystrings)) {   # loop over the length of mystrings
  print(switch(  # overwrite every position of mynums
    EXPR = mystrings[i],   # compare string value to the list string values
    Homer=12,
    Marge=34,
    Bart=56,
    Lisa=78,
    Maggie=90,
    NA))
}
```

```
[1] NA
[1] 12
[1] NA
[1] NA
[1] 90
[1] 56
```