The basics: 07 loops

Ari Anisfeld

9/8/2020

Questions

Recall, for-loops are an iterator that help us repeat tasks while changing inputs. The most common structure for your code will look like the following code. This can be simplified if you are not storing results.

```
# what are you iterating over? The vector from -10:10
items_to_iterate_over <- c(-10:10)

# pre-allocate the results
out <- rep(0, length(items_to_iterate_over))

# write the iteration statement --
# we'll use indices so we can store the output easily
for (i in seq_along(items_to_iterate_over)) {

  # do something
  # we capture the median of three random numbers
  # from normal distributions various means
  out[[i]] <- median(rnorm(n = 3, mean = items_to_iterate_over[[i]]))

}
```

Writing for-loops

1. Write a for-loop that prints the numbers `5, 10, 15, 20, 250000`.
2. Write a for-loop that iterates over the indices of `x` and prints the `i`th value of `x`.

```
x <- c(5, 10, 15, 20, 250000)

# replace the ... with the relevant code

for (i in ... ){
  print(x[[...]])
}
```

3. Write a for-loop that simplifies the following code so that you don't repeat yourself! Don't worry about storing the output yet. Use `print()` so that you can see the output. What happens if you don't use `print()`?

```
sd(rnorm(5))
sd(rnorm(10))
sd(rnorm(15))
```

```
sd(rnorm(20))
sd(rnorm(25000))
```

a. adjust your for-loop to see how the `sd` changes when you use `rnorm(n, mean = 4)`
b. adjust your for-loop to see how the `sd` changes when you use `rnorm(n, sd = 4)`

1. Now store the results of your for-loop above in a vector. Pre-allocate a vector of lenth 5 to capture the standard deviations.

vectorization vs for loops

Recall, vectorized functions operate on a vector item by item. It's like looping over the vector!

The following for-loop is better written vectorized.

Compare the loop version

```
names <- c("Alysha", "Fanmei", "Paola")

out <- character(length(names))

for (i in seq_along(names)) {
  out[[i]] <- paste0("Welcome ", names[[i]])
}
```

to the vectorized version

```
names <- c("Alysha", "Fanmei", "Paola")
out <- paste0("Welcome ", names)
```

The vectorized code is preferred because it is easier to write and read, and is possibly more efficient.[1]

1. Rewrite your first for-loop, where you printed `5, 10, 15, 20, 250000` as vectorised code
2. Rewrite this for-loop as vectorized code:

```
radii <- c(0:10)

area <- double(length(radii))

for (i in seq_along(radii)) {

  area[[i]] <- pi * radii[[i]] ^ 2

}
```

3. Rewrite this for-loop as vectorized code:

```
radii <- c(-1:10)

area <- double(length(radii))

for (i in seq_along(radii)) {

  if (radii[[i]] < 0) {
    area[[i]] <- NaN
  } else {
```

[1] In this case, I had about a 10 times speed up when I benchmarked the code, but I'm not sure if that's real.

```
    area[[i]] <- pi * radii[[i]] ^ 2
  }

}
```

Solutions

Writing for-loops

1. Write a for-loop that prints the numbers `5, 10, 15, 20, 250000`.

```
 x <- c(5, 10, 15, 20, 250000)

  # replace the ... with the relevant code

for (number in x){
  print(number)
}
```

```
## [1] 5
## [1] 10
## [1] 15
## [1] 20
## [1] 250000
```

2. Write a for-loop that iterates over the indices of `x` and prints the `i`th value of `x`.

```
x <- c(5, 10, 15, 20, 250000)

# replace the ... with the relevant code

for (i in seq_along(x) ){
  print(x[[i]])
}
```

```
## [1] 5
## [1] 10
## [1] 15
## [1] 20
## [1] 250000
```

3. Write a for-loop that simplifies the following code so that you don't repeat yourself!

```
sd(rnorm(5))
sd(rnorm(10))
sd(rnorm(15))
sd(rnorm(20))
sd(rnorm(25000))
```

a. adjust your for-loop to see how the `sd` changes when you use `rnorm(n, mean = 4)`

b. adjust your for-loop to see how the `sd` changes when you use `rnorm(n, sd = 4)`

```
x <- c(5, 10, 15, 20, 250000)

# replace the ... with the relevant code
```

```
for (i in seq_along(x) ){
  print(sd(rnorm(x[[i]])))
}
```

```
## [1] 0.6376168
## [1] 0.6904605
## [1] 1.015602
## [1] 1.047319
## [1] 0.999501
```

```
for (i in seq_along(x) ){
   n <- x[[i]]
  print(sd(rnorm(n, mean = 4)))
}
```

```
## [1] 1.17934
## [1] 0.9886478
## [1] 0.9194763
## [1] 0.9382895
## [1] 1.002549
```

```
for (i in seq_along(x) ){
  n <- x[[i]]
  print(sd(rnorm(n, sd = 4)))
}
```

```
## [1] 3.958886
## [1] 3.89497
## [1] 4.192581
## [1] 2.869024
## [1] 4.002512
```

This might be a time to write function!

```
print_sds <- function(x, mean = 0, sd = 1) {
  for (i in seq_along(x) ){
    # I add the next two lines for clarity
    n <- x[[i]]
    sample <- rnorm(n, mean = mean, sd = sd)
    print(sd(sample))
  }
}

print_sds(x)
```

```
## [1] 0.7495351
## [1] 0.6739105
## [1] 0.7905733
## [1] 0.9873447
## [1] 0.9987092
```

```
print_sds(x, mean = 4)
```

```
## [1] 0.9325725
## [1] 1.017511
## [1] 0.8991141
## [1] 0.924617
```

```
## [1] 0.9980651
```

```
print_sds(x, sd = 4)
```

```
## [1] 5.627013
## [1] 2.530572
## [1] 3.483696
## [1] 5.83558
## [1] 4.005663
```

1. Now store the results of the first for-loop you wrote above. Pre-allocate a vector of lenth 5 to capture the standard deviations.

```
x <- c(5, 10, 15, 20, 250000)

sd_of_samples <- rep(0, length(x))

for (i in seq_along(x) ){

  sd_of_samples[[i]] <- sd(rnorm(x[[i]]))

}
```

vectorization vs for loops

1. Rewrite your first for-loop, where you printed `5, 10, 15, 20, 250000` as vectorised code

```
x <- c(5, 10, 15, 20, 250000)
print(x)
```

```
## [1]      5     10     15     20 250000
```

2. Rewrite this for-loop as vectorized code:

```
radii <- c(0:10)

area <- pi * radii ^ 2
```

3. Rewrite this for-loop as vectorized code:

```
radii <- c(-1:10)

area <- ifelse(radii >= 0, pi * radii ^ 2, NaN)
```