Week-6: Iterations

NM2207: Computational Media Literacy

Narayani Vedam, Ph.D.

Department of Communications and New Media

NUS National University of Singapore | Faculty of Arts & Social Sciences

This week

Table of contents


```
# Example of a for loop

for(value in c(1, 2, 3, 4, 5)) {

  print(value)

  }
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

I. `for` loops

Printing elements of a vector: `c(3,6,9)`

```
# Method 1: repeating the code
x<-c(3,6,9)
print(x[1])
```

```
## [1] 3
```

```
print(x[2])
```

```
## [1] 6
```

```
print(x[3])
```

```
## [1] 9
```

```
# Method 2: function to print the value of th
print_function <- function(x) print(x)
# Function call
print_function(3)
```

```
## [1] 3
```

```
print_function(6)
```

```
## [1] 6
```

```
print_function(9)
```

```
## [1] 9
```

Simple `for` loop

```
for (x in c(3, 6, 9)) {

  print(x)

}
```

```
## [1] 3
## [1] 6
## [1] 9
```

What is goin on?

- Each value in `c(3,6,9)` is assigned to `x`
- The action occurs between the curly brackets

`for` loops: structure

The general structure of a `for` loop is as follows,

```
for (value in list_of_values)
  {do something (based on value)}
```

```
for (index in list_of_indices)
  {do something (based on index)}
```

```
for (x in 1:8) {print(x)}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
```

```
for (x in 1:8)
  {y <- seq(from=100,to=200,by=5)
  print(y[x])}
```

```
## [1] 100
## [1] 105
## [1] 110
## [1] 115
## [1] 120
## [1] 125
## [1] 130
## [1] 135
```

Example: find sample means

Suppose we want to find the means of increasingly large samples.

```
mean1 <- mean(rnorm(5))
mean2 <- mean(rnorm(10))
mean3 <- mean(rnorm(15))
mean4 <- mean(rnorm(20))
mean5 <- mean(rnorm(25000))
print(c(mean1, mean2, mean3, mean4, mean5))
```

```
## [1] -0.207678888  0.137153323  0.201175393  0.105456183 -0.003075514
```

Example: find sample means

Let's avoid repeating code with a for loop

```
# 1. determine what to loop over
sample_sizes <- c(5, 10, 15, 20, 25000)
# 2. pre-allocate space to store output
sample_means <- double(length(sample_sizes))

for (i in seq_along(sample_sizes)) {

  sample_means[i] <- mean(rnorm(sample_sizes[i]))

}

sample_means
```

```
## [1] -0.1804937002  0.1559834824  0.2748488056  0.4884687113 -0.0003086708
```

In the subsequent slides we'll explain each step.

Finding sample means, broken down

Initialization of variables before starting the `for` loop:

1. Create a vector of the `sample_sizes` we want to use
2. Create a vector to store the output

```
# 1. determine what to loop over
sample_sizes <- c(5, 10, 15, 20, 25000)

# 2. pre-allocate space to store output
sample_means <- double(length(sample_sizes))
```

What does `sample_means` currently look like?

```
# 2. pre-allocate space to store output
sample_means <- double(length(sample_sizes))
```

Why do this?

- It makes the code more efficient
- An alternative is to build up an object as you go
- But, that would mean copying the data over and over again

Alternative ways to preallocate space

```r
# Example 1 for data_type=double
sample_means <- vector("double", length = 5)

# Example 2 for data_type=double
sample_means <- double(5)

# Example 3 for data_type=double
sample_means <- rep(0, length(sample_sizes))
```

1. Each data type has a corresponding function,
    - `logical()`
    - `integer()`
    - `character()`
2. To hold data of different types, use `lists`

```r
data_list <- vector("list", length = 5)
```

Adding data to a vector, broken down

Determine what sequence to loop over.

```
for (i in 1:length(sample_sizes)) {
}
```

A helper function `seq_along()`

`seq_along(x)` is synonymous to `1:length(x)` where `x` is a vector.

```
# Declare and initialize a vector
vec <- c("x", "y", "z")

# Generate indices as many as the length of the vector
1:length(vec)
```

```
## [1] 1 2 3
```

```
# Instead use seq_along to generate indices as many as the length of the vector
seq_along(vec)
```

```
## [1] 1 2 3
```

A helper function `seq_along()`

```r
# Another example with a vector sample_sizes
sample_sizes <- c(5, 10, 15, 20, 25000)

# Generate indices using seq_along
seq_along(sample_sizes)
```

```
## [1] 1 2 3 4 5
```

What if `sample_sizes` is accidentally a 0-length vector? Can you try?

Adding data to a vector

```r
# Initialize a vector
sample_sizes <- c(5, 10, 15, 20, 25000)

# Initialize a vector of zeros of the same size as sample_sizes
sample_means <- rep(0, length(sample_sizes))

# # Another way to initialize a vector of zeros is here below
# sample_means <- numeric(length(sample_sizes))

for (i in seq_along(sample_sizes)) {

  sample_means[i] <- mean(rnorm(sample_sizes[i]))

}
```

Save the mean of the sample to the ith place of the `sample_means` vector

Common Error

```r
# Initialize a vector
sample_sizes <- c(5, 10, 15, 20, 25000)

# Initialize a vector of zeros of the same size as sample_sizes
sample_means <- rep(0, length(sample_sizes))

for (i in seq_along(sample_sizes)) {
  mean(rnorm(sample_sizes[i]))
}

sample_means
```

```
## [1] 0 0 0 0 0
```

- This code fails, because we do not store the output in `sample_means` in the `for` loop!
- We're calculating the mean, but it's not being saved anywhere

Review: Vectorized operations

When possible, take advantage of the fact that R is vectorized.

```r
# Example: bad idea!
# Vector with numbers from 7 to 11
a <- 7:11
# Vector with numbers from 8 to 12
b <- 8:12
# Vector of all zeros of length 5
out <- rep(0L, 5)
# Loop along the length of vector a
for (i in seq_along(a)) {
  # Each entry of out is the sum of the corre
  out[i] <- a[i] + b[i]
}
out
```

```
## [1] 15 17 19 21 23
```

```r
# Taking advantage of vectorization
# Vector with numbers from 7 to 11
a <- 7:11
# Vector with numbers from 8 to 12
b <- 8:12
out <- a + b
out
```

```
## [1] 15 17 19 21 23
```

Use vectorized operations and `tidyverse` functions like `mutate()` whenever you can

II. Functionals

`for` loops vs. functionals

Rewriting the earlier example by wrapping it in a function,

```
# Initialise a vector with the size of 5 different samples
sample_sizes <- c(5, 10, 15, 20, 25000)

# Define a function that wraps the for loop
fmean <- function(sample_sizes){

  # Initialize an empty vector of the same length as sample_sizes
  sample_means <- rep(0, length(sample_sizes))

  # Compute the mean of each sample
  for (i in seq_along(sample_sizes)) {

    sample_means[i] <- mean(rnorm(sample_sizes[i]))

  }

}
```

As an after-thought, you want to compute the median and the standard deviation

`for` loops vs. functionals

To do so, you copy and paste `fmean()` function and replace `mean()` with `median()`

```
# Initialise a vector with the size of 5 different samples
sample_sizes <- c(5, 10, 15, 20, 25000)

# Function to compute the median
fmedian <- function(sample_sizes){

  # Initialize an empty vector of the same length as sample_sizes
  sample_medians <- rep(0, length(sample_sizes))

  # Compute the median of each sample
  for (i in seq_along(sample_sizes)) {

    sample_medians[i] <- median(rnorm(sample_sizes[i]))

  }

}
```

`for` loops vs. functionals

To do so, you copy and paste `fmean()` function and replace `mean()` with `sd()`

```
# Initialise a vector with the size of 5 different samples
sample_sizes <- c(5, 10, 15, 20, 25000)

# Function to compute the standard deviation
fsd <- function(sample_sizes){

  # Initialize an empty vector of the same length as sample_sizes
  sample_sds <- rep(0, length(sample_sizes))

  # Compute the sd of each sample
  for (i in seq_along(sample_sizes)) {

    sample_sds[i] <- sd(rnorm(sample_sizes[i]))

  }

}
```

Uh oh! You've copied-and-pasted this code twice

`for` loops vs Functionals

- Is there a way to generalise the function in a way that it can compute, `mean()`, `median()` and `sd()`??
- In the three functions, `fmean()`, `fmedian()` and `fsd()` only one thing changed!
- The functions inside the `for` loop - `mean()`, `median()` and `sd()`

```
# Initialise a vector with the size of 5 different samples
sample_sizes <- c(5, 10, 15, 20, 25000)
# Create a functional- function inside a function
sample_summary <- function(sample_sizes, fun) {
# Initialise a vector of the same size as sample_sizes
out <- vector("double", length(sample_sizes))
# Run the for loop for as long as the length of sample_sizes
for (i in seq_along(sample_sizes)) {
    # Perform operations indicated fun
    out[i] <- fun(rnorm(sample_sizes[i]))
  }
return(out)
}
```

`for` loops vs Functionals

- Do you notice a function `fun` is passed to another `sample_summary`
- Such nested functions are called ***functionals***, and are unique to R

```
# Computing mean
sample_summary(sample_sizes,mean)
```

```
## [1] -0.475924601 -0.694917610  0.328722971  0.138877248 -0.001374265
```

```
# Computing median
sample_summary(sample_sizes,median)
```

```
## [1]  0.26531301 -0.21325005 -0.14660619  0.05722821  0.01484544
```

```
# Computing sd
sample_summary(sample_sizes,sd)
```

III. `while` loops

`while` loop

- It can be used when the length of the input sequence is unknown
- It is more general than a for loop: you can rewrite any for loop as a while loop but not the other way around

```
# General structure
while(condition){
  # body of the while loop
}
```

```
# Example
#for loop
for (i in seq_along(x)) {
  # body
}
# Equivalent to
i <- 1
while (i <= length(x)) {
  # body
  i <- i + 1
}
```

`while` loop: example

```r
# General structure
for(i in 1:5){
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

```r
# Example
i <- 1
while (i <= 5) {
  # body
  print(i)
  i <- i + 1
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

Thanks!

Slides created via the R packages:

xaringan
gadenbuie/xaringanthemer.

Faculty of Arts
& Social Sciences