Week-6: Code-along

WU YUTONG NM2207: Computational Media Literacy

2023-09-19

II. Code to edit and execute using the Code-along-6.Rmd file

A. `for` loop

1. Simple `for` loop (Slide #6)

```
# Enter code here
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✔ dplyr     1.1.2     ✔ readr     2.1.4
## ✔ forcats   1.0.0     ✔ stringr   1.5.0
## ✔ ggplot2   3.4.3     ✔ tibble    3.2.1
## ✔ lubridate 1.9.2     ✔ tidyr     1.3.0
## ✔ purrr     1.0.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

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

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

2. `for` loops structure (Slide #7)

```
# Left-hand side code: for loop for passing values
for(x in 1:8) {print(x)}
```

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

```
# Right-hand side code: for loop for passing indices
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
```

3. Example: find sample means (Slide #9)

```
# Enter code here
sample_sizes <- c(5,10, 15, 20, 25000)
sample_means <- double(length(sample_sizes))

for(i in seq_along(sample_sizes)) {sample_means[[i]] <- mean(rnorm(sample_sizes[[i]]))
}
sample_means
```

```
## [1]  1.212967593 -0.235871548  0.091843869  0.143317285  0.003210293
```

4. Alternate ways to pre-allocate space (Slide #12)

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

```
# Initialisation of data_list
sample_sizes <- c(5, 10, 15, 20, 25000)
sample_means <- rep(0, length(sample_sizes))
for (i in seq_along(sample_sizes)){
  sample_means[[i]] <- mean(rnorm(sample_sizes[[i]]))
}
sample_means
```

```
## [1] 0.313258791 0.730361424 0.068476509 0.178203315 0.002152797
```

5. Review: Vectorized operations (Slide #18)

```
# Example: bad idea!
a <- 7:11
b <- 8:12
out <- rep(0L, 5)
for (i in seq_along(a)) {
  out[i] <- a[i] + b[i]
}
out
```

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

```
# Taking advantage of vectorization
a <- 7:11
b <- 8:12
out <- a + b
out
```

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

B. Functionals

6. `for` loops vs Functionals (Slides #23 and #24)

```
# Slide 23
# 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)
}
```

```
# Slide 24
#Compute mean
sample_summary(sample_sizes,mean)
```

```
## [1] -0.0838753669 -0.4996994014 -0.1778876826 -0.2987801378 -0.0009686679
```

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

```
## [1] -1.223908799  0.142563858  0.263959696 -0.558461858  0.001872121
```

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

```
## [1] 0.6297982 0.9648637 1.2531264 0.9672324 1.0069288
```

C. `while` loop

7. `while` loop (Slides #27)

```
# Left-hand side code: for loop
for(i in 1:5) {
  print(i)
}
```

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

```
# Right-hand side code: while loop
i <- 1
while(i <= 5) {
  print(i)
  i <- i +1
}
```

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