Week-6: Code-along

Tan Ee Xuan

2023-09-16

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
for(x in c(2,4,5,1)) {
  print(x)
  }
```

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

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 <- c(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
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.2 ──
## ✔ ggplot2 3.3.6     ✔ purrr   0.3.4
## ✔ tibble  3.1.8     ✔ dplyr   1.0.9
## ✔ tidyr   1.2.0     ✔ stringr 1.4.0
## ✔ readr   2.1.2     ✔ forcats 0.5.1
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
```

```
sample_sizes <- c(12,43,15,76,21)

sample_means <- double(length(sample_sizes))

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

sample_means
```

```
## [1] -0.01551644 -0.25503625 -0.52685865  0.12256338 -0.12423675
```

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

```
# Example 3 for data_type=double
sample_means <- vector("double", length =5)
sample_means <- double(5)
sample_means <- rep(0, length(sample_sizes))
```

```
# Initialisation of data_list
data_list <- vector("list", length = 5)

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

5. Review: Vectorized operations (Slide #18)

```
# Example: bad idea!
a <- 7:11
b <- 8:12

out <- rep(0L, 5)

seq_along(a)
```

```
## [1] 1 2 3 4 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
sample_sizes <- c(5,10,15,20,25000)

sample_summary <- function(sample_sizes, fun) {
  out <- vector("double", length(sample_sizes))
  for (i in seq_along(sample_sizes)) {
    out[i] <- fun(rnorm(sample_sizes[i]))
  }
  return(out)
}
```

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

```
## [1] -0.843325675  0.174310264 -0.177192550 -0.004304898  0.007028424
```

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

```
## [1]  0.34977823  0.04427973  0.29397283 -0.06739937  0.01060958
```

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

```
## [1] 0.5033736 1.1409374 1.0041343 0.7188278 0.9984696
```

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
```