Week-6: Code-along

Dawn Cheung

2023-09-15

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(3, 6, 9)) { #x assumes the value of each element in the vector, and gets printed accordingly
  print(x)
}
```

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

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

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

```
# 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
```

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

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

y <- seq(from=100, to=200, by=5) #hence u get vector y

for (x in 1:8) {
  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)

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

# 2. pre-allocate space to store output (for efficiency)
sample_means <- double(length(sample_sizes))
# for alternatives, see part 4 of code along

for (i in seq_along(sample_sizes)) { #seq_along is like the indexes of the vector. so seq_alo
ng(sample_sizes) == 1:length(sample_sizes). so you get consecutive numbers.
  sample_means[[i]] <- mean(rnorm(sample_sizes[[i]])) #slay
}
sample_means
```

```
## [1]  0.252314546 -0.338887369  0.097372125 -0.166970402 -0.006667334
```

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

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

```r
# Initialisation of data_list
#to hold data of different types
data_list <- vector("list", length = 5)
```

5. Review: Vectorized operations (Slide #18)

```
# 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)

for (i in seq_along(a)) {
  out[i] <- a[i] + b[i]
}
```

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

out <- a + b
out
```

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

```
#since a and b are vectors of the same length, we dont need to iteratively add them
```

B. Functionals

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

no section for slide 21 [rewrite of earlier example by wrapping it in a function]

```
# Slide 23
sample_sizes <- c(5, 10, 15, 20, 25000)

fsd <- function(sample_sizes) {

  #initalize  an empty vector of the same lenth 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

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])) #so ur parameter of 'fun' can become the function u
sed here (unique to R, how sad)
    }
  return(out)
}
```

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

```
## [1]  0.08835237  0.02943609  0.36889740 -0.48087393 -0.00810464
```

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

```
## [1]  0.156081382  0.300055823  0.385694501 -0.091343262  0.004933907
```

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

```
## [1] 1.2032222 0.9020758 1.0705193 0.8537851 1.0038079
```

C. `while` loop

7. `while` loop (Slides #27)

while(condition) {} you can always write a for loop as a while loop, but you cannot write all while loops as for loops

```r
#SLIDE TWENTY-NINE (29)
# Left-hand side code: for loop
for (i in 1:5) {
  print(i)
}
```

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

```r
# 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
```