Ch7. Programming strunctures

Jongrak Jeong

2023-01-16

1. Controling 'loop'

for (name in expr_1) expr_2

name: R variable expr_1 and expr_2: R expression.

ex1. from 1 to 10, summation of i^4?

$$\sum_{i=1}^{10} i^4$$

```
temp <- 0
for (i in 1:10) temp <- temp + i ^ 4
temp
```

```
## [1] 25333
```

ex2. for x = 10, 7, 4, and 1, summation of x^(1/2)?

```
temp <- 0
x <- 10
for (n in 1:4) {
  print(x)
  temp <- temp + sqrt(x)
  x <- x - 3
}
```

```
## [1] 10
## [1] 7
## [1] 4
## [1] 1
```

```
temp
```

```
## [1] 8.808029
```

while (condition) expr

ex3. from i = 1 to i ^ 4 < 100000, summation of i ^ 4?

$$\sum_{i \geq 1\,,\ i^4 < 100000} i^4$$

```r
temp <- 0
i <- 1
while (i^4 < 100000) {
  temp <- temp + i^4
  i <- i + 1
  }
c(i-1, temp)
```

```
## [1]     17 327369
```

repeat{} and break: repeat something until it meets 'break'

ex3. (continued)

```r
temp <- 0
i <- 1
repeat {
  if (i^4 >= 100000) break
  temp <- temp + i^4
  i <- i+1
}
c(i-1, temp)
```

```
## [1]     17 327369
```

2. if-else

if and else

if (condition) statement1 else statement2

ex1. [baseball] hit counting in the first inning (Kids baseball) : Monte-Carlo script

- hit or not
- 3 outs, then over
- prob. of hit: 0.4 and prob. of out: 0.6

```r
outs <- 0
hits <- 0
repeat {
  x <- sample(c(0, 1), 1, prob = c(0.6, 0.4))
  print(x)
  if (x == 1) hits <- hits + 1 else outs <- outs + 1
  if (outs >= 3) break
}
```

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

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

```
hits
```

```
## [1] 11
```

ex2. [gambling] bet $1 each game and game over if someone lose everything

- starts with $3
- prob. of getting 1: 0.4, prob. of losing 1: 0.6

```
position <- 3
count <- 1
repeat {
  x <- sample(c(0, 1), 1, prob = c(0.6, 0.4))
  print(x)
  position <- if (x == 1) position + 1 else position - 1
  if (position <= 0) break
  count <- count + 1
}
```

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

```
count
```

```
## [1] 13
```

```
position <- 3
count <- 1
repeat {
  x <- sample(c(0, 1), 1, prob = c(0.6, 0.4))
  print(x)
  position <- position + ifelse (x == 1, 1, -1)
  if (position <= 0) break
  count <- count + 1
}
```

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

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

```
count
```

```
## [1] 17
```

if-else vs ifelse

```
cond <- sample(c("T", "F"), 8, replace = T)
cond
```

```
## [1] "T" "T" "T" "T" "F" "F" "F" "F"
```

```
score <- vector(length = 8)
score <- ifelse (cond, "win", "lose")
score
```

```
## [1] "win"  "win"  "win"  "win"  "lose" "lose" "lose" "lose"
```

3. user-defined functions

f <- function(arguments) { statements :

}

ex1. [kids baseball]

```
baseball <- function(prob) {
  outs <- 0; hits <- 0
  repeat {
    x <- sample(c(0, 1), 1, prob = prob)
    if (x == 1) hits <- hits + 1 else outs <- outs + 1
    if (outs >= 3) break
  }
  return (hits)
}

baseball(c(0.75, 0.25))
```

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

ex1. total hits

```r
game.hits <- 0
for (i in 1:9) game.hits <- game.hits + baseball(c(0.75, 0.25))
game.hits
```

```
## [1] 12
```

```r
# outs: local variable > error
```

```r
# make 'outs' variable global variable
baseball.1 <- function(prob) {
  outs <<- 0; hits <- 0
  repeat {
    x <- sample(c(0, 1), 1, prob = prob)
    if (x == 1) hits <- hits + 1 else outs <<- outs + 1
    if (outs >= 3) break
  }
  return (hits)
}

baseball.1(c(0.75, 0.25))
```

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

```r
outs
```

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

return vectors of more than two: return list()

```r
baseball <- function(prob) {
  outs <- 0
  hits <- 0

  repeat {
    x <- sample(c(0, 1), 1, prob = prob)
    if (x == 1) hits <- hits + 1 else outs <- outs + 1
    if (outs >= 3) break
  }

  return(list(hits = hits, outs = outs))
}

baseball(c(0.75, 0.25))
```

```
## $hits
## [1] 0
##
## $outs
## [1] 3
```

4. application of the functions

ex1. baseball: winning score of one inning

```r
one.inning <- function (prob = c(0.75, 0.15, 0.05, 0.025, 0.025)) {
  run <- 0
  out <- 0
  base <- c(0, 0, 0, 0, 0, 0, 0)

  repeat {
    hit <- sample(0:4, 1, prob = prob)
    if (hit > 0) {
      base[hit + (1:3)] <- base[1:3]
      base[hit] <- 1
      if (hit > 1) base[1:(hit - 1)] <- 0
      run <- run + sum(base[4:7])
      base[4:7] <- 0
    } else {
      out <- out + 1
      if (out >= 3) break
    }
  }

  return (list(run = run, out = out, bases = base[1:3]))
}

# Check
prob.A <- c(0.75, 0.15, 0.05, 0.025, 0.025)
prob.B <- c(0.75, 0.15, 0.05, 0.025, 0.025)
score.A <- 0
score.B <- 0

for (i in 1:9) {
  score.A <- score.A + one.inning(prob.A)$run
  score.B <- score.A + one.inning(prob.B)$run
}
c(score.A, score.B, if(score.A > score.B) "A" else if(score.A == score.B) "draw" else "B")
```

```
## [1] "2"    "2"    "draw"
```

ex2. quick sort - recursive calls of a function

sort numeric vector with length n. 1) key vs therest, sv1(left) <- smaller one and sv(right) <- bigger one 2) for sv1 and sv2, 1) repeated

```r
# define quick sort function
quick <- function(x) {
  if (length(x) <= 1) return(x)
  pivot <- x[1]
  therest <- x[-1]
  sv1 <- therest[therest < pivot]
  sv2 <- therest[therest >= pivot]
  sv1 <- quick(sv1)
  sv2 <- quick(sv2)
  return(c(sv1, pivot, sv2))
}

# check
```

```r
x <- round(runif(100), 2)

x
```

```
##   [1] 0.48 0.92 0.05 0.93 0.97 0.23 0.53 0.52 0.52 0.41 0.50 0.68 0.25 0.82 0.28
##  [16] 0.26 0.10 0.85 0.22 0.71 0.40 0.09 0.21 0.19 0.15 0.08 0.88 0.94 0.21 0.75
##  [31] 0.97 0.53 0.91 0.59 0.04 0.97 0.05 0.74 0.96 0.12 0.95 0.29 0.44 0.08 0.46
##  [46] 0.26 0.14 0.13 0.99 0.69 0.95 0.40 0.54 0.49 0.61 0.43 0.65 0.78 0.66 0.64
##  [61] 0.62 0.37 0.99 0.75 0.66 0.15 0.84 0.61 0.06 0.01 0.45 0.34 0.48 0.26 0.73
##  [76] 0.08 0.18 0.90 0.91 0.42 0.27 0.62 0.43 0.31 0.69 0.80 0.98 0.87 0.68 0.67
##  [91] 0.40 0.90 0.98 0.82 0.68 0.83 0.74 0.03 0.89 0.98
```

```r
quick(x)
```

```
##   [1] 0.01 0.03 0.04 0.05 0.05 0.06 0.08 0.08 0.08 0.09 0.10 0.12 0.13 0.14 0.15
##  [16] 0.15 0.18 0.19 0.21 0.21 0.22 0.23 0.25 0.26 0.26 0.26 0.27 0.28 0.29 0.31
##  [31] 0.34 0.37 0.40 0.40 0.40 0.41 0.42 0.43 0.43 0.44 0.45 0.46 0.48 0.48 0.49
##  [46] 0.50 0.52 0.52 0.53 0.53 0.54 0.59 0.61 0.61 0.62 0.62 0.64 0.65 0.66 0.66
##  [61] 0.67 0.68 0.68 0.68 0.69 0.69 0.71 0.73 0.74 0.74 0.75 0.75 0.78 0.80 0.82
##  [76] 0.82 0.83 0.84 0.85 0.87 0.88 0.89 0.90 0.90 0.91 0.91 0.92 0.93 0.94 0.95
##  [91] 0.95 0.96 0.97 0.97 0.97 0.98 0.98 0.98 0.99 0.99
```

ex3. binary search

```r
search <- function(compare, x) {
  n <- length(compare)
  n.1 <- 1; n.2 <- n

  if (x <= min(compare)) print("x is not larger than the minimum")
  if (x >= max(compare)) print("x is not smaller than the maximum")
  if (x > min(compare) & x < max(compare)) {
    repeat {
      if (n.2 - n.1 <= 1) break
      k <- round((n.1 + n.2) / 2)
      if(compare[k] < x) n.1 <- k else n.2 <- k
    }
  }

  return(c(n.1, n.2))
}

score <- sort(rnorm(100, 50, 10))
z <- rnorm(1, 50, 10)
score
```

```
##   [1] 30.50573 30.55950 33.90455 34.10445 35.75479 35.96995 36.46791 39.42986
##   [9] 39.57528 39.92631 39.95315 40.09881 40.49260 40.82290 40.87081 41.17017
##  [17] 41.60650 41.66943 42.14170 42.30236 43.06017 43.34180 43.88593 44.05616
##  [25] 44.14620 44.35190 44.35583 44.51280 44.73263 44.75800 44.82543 45.43185
##  [33] 45.45694 45.69580 45.82805 46.04804 46.34478 46.56015 46.90292 47.02139
##  [41] 47.10329 47.49127 47.62955 47.97569 48.12661 48.30326 49.70680 49.77194
##  [49] 51.12133 51.12679 51.14597 51.20536 51.28629 51.75641 53.32131 53.38833
##  [57] 53.52896 53.84332 54.04161 54.07488 54.15113 54.85656 54.93437 54.96633
##  [65] 55.26575 55.46617 55.55378 55.63430 56.27858 56.36106 56.45489 56.55041
##  [73] 57.07758 57.08025 57.35584 57.37246 57.96040 58.31209 58.35239 58.57453
```

```
##  [81] 58.95005 59.27483 59.79305 60.29414 61.11276 61.43984 61.79549 61.90036
##  [89] 62.28518 64.90325 65.25019 66.72512 66.84568 67.94061 69.28698 69.98933
##  [97] 70.57566 73.51202 74.59204 83.34019
```

```
z
```

```
## [1] 84.50746
```

```
index <- search(score, z)
```

```
## [1] "x is not smaller than the maximum"
```

```
round(c(left = score[index[1]], x = z, right = score[index[2]]), 1)
```

```
##  left     x right
##  30.5  84.5  83.3
```

ex4. blackjack

```
cards <- sample(rep(c("A", 2:10, "J", "Q", "K"), 4))

score.1 <- function(x) {
  if (x %in% c("J", "Q", "K")) return(10)
  else if (x == "A") return(11)
  else return(as.numeric(x))
}

score.2 <- function(x) {
  if (x %in% c("J", "Q", "K")) return(10)
  else if (x == "A") return(1)
  else return(as.numeric(x))
}

d.cards <- cards[1:2]
p.cards <- cards[3:4]
count <- 4

d.score <- sum(sapply(d.cards, score.1))
p.score <- sum(sapply(p.cards, score.1))

while (p.score < 14) {
  p.cards <- c(p.cards, cards[count + 1])
  p.score <- p.score + score.2(cards[count + 1])
  count <- count + 1
}
p.bust <- ifelse(p.score >= 22, 1, 0)

while (d.score < 17) {
  p.cards <- c(d.cards, cards[count + 1])
  d.score <- d.score + score.2(cards[count + 1])
  count <- count + 1
}
d.bust <- ifelse(d.score >= 22, 1, 0)

if(p.bust == 1) result <- "lose" else
  if(d.bust == 1) result <- "win" else
```

```
    if(p.score > d.score) result <- "win" else
      if(p.score < d.score) result <- "lose" else
        result <- "draw"

result
```

```
## [1] "lose"
```

```
p.cards
```

```
## [1] "2" "8" "A" "A" "2"
```

```
ifelse(p.bust == 1, "player bust", p.score)
```

```
## [1] 14
```

```
d.cards
```

```
## [1] "9" "J"
```

```
ifelse(d.bust == 1, "dealer bust", d.score)
```

```
## [1] 19
```

practice problem 1-1.

```
result <- vector(length = 1000)

for(i in 1:1000) {
cards <- sample(rep(c("A", 2:10, "J", "Q", "K"), 4))

score.1 <- function(x) {
  if (x %in% c("J", "Q", "K")) return(10)
  else if (x == "A") return(11)
  else return(as.numeric(x))
}

score.2 <- function(x) {
  if (x %in% c("J", "Q", "K")) return(10)
  else if (x == "A") return(1)
  else return(as.numeric(x))
}

d.cards <- cards[1:2]
p.cards <- cards[3:4]
count <- 4

d.score <- sum(sapply(d.cards, score.1))
p.score <- sum(sapply(p.cards, score.1))

while (p.score < 14) {
  p.cards <- c(p.cards, cards[count + 1])
  p.score <- p.score + score.2(cards[count + 1])
  count <- count + 1
}
p.bust <- ifelse(p.score >= 22, 1, 0)
```

```
while (d.score < 17) {
  p.cards <- c(d.cards, cards[count + 1])
  d.score <- d.score + score.2(cards[count + 1])
  count <- count + 1
}
d.bust <- ifelse(d.score >= 22, 1, 0)

if(p.bust == 1) result[i] <- "lose" else
  if(d.bust == 1) result[i] <- "win" else
    if(p.score > d.score) result[i] <- "win" else
      if(p.score < d.score) result[i] <- "lose" else
        result[i] <- "draw"
}

result_tab <- table(result)
result_tab
```

```
## result
## draw lose  win
##   73  508  419
```

practice problem 1-2. 14 -> 13

```
result <- vector(length = 1000)

for(i in 1:1000) {
cards <- sample(rep(c("A", 2:10, "J", "Q", "K"), 4))

score.1 <- function(x) {
  if (x %in% c("J", "Q", "K")) return(10)
  else if (x == "A") return(11)
  else return(as.numeric(x))
}

score.2 <- function(x) {
  if (x %in% c("J", "Q", "K")) return(10)
  else if (x == "A") return(1)
  else return(as.numeric(x))
}

d.cards <- cards[1:2]
p.cards <- cards[3:4]
count <- 4

d.score <- sum(sapply(d.cards, score.1))
p.score <- sum(sapply(p.cards, score.1))

while (p.score < 13) {
  p.cards <- c(p.cards, cards[count + 1])
  p.score <- p.score + score.2(cards[count + 1])
  count <- count + 1
}
p.bust <- ifelse(p.score >= 22, 1, 0)
```

```
while (d.score < 17) {
  p.cards <- c(d.cards, cards[count + 1])
  d.score <- d.score + score.2(cards[count + 1])
  count <- count + 1
}
d.bust <- ifelse(d.score >= 22, 1, 0)

if(p.bust == 1) result[i] <- "lose" else
  if(d.bust == 1) result[i] <- "win" else
    if(p.score > d.score) result[i] <- "win" else
      if(p.score < d.score) result[i] <- "lose" else
        result[i] <- "draw"
}

result_tab <- table(result)
result_tab
```

```
## result
## draw lose  win
##   74  478  448
```