

R_Control Structures and Functions R

Malik Haroon

Advanced Statistical Programming (ACCA7022)

```

> #=====
> # ACTIVITY 1: Classify Numbers as Positive, Negative, or Zero
> #=====
>
> cat("=====\n")
=====
> cat("ACTIVITY 1: Number Classifier\n")
ACTIVITY 1: Number Classifier
> cat("=====\n\n")
=====

>
> # Function to classify a single number
> classify_number <- function(x) {
+   if (x > 0) {
+     return(paste(x, "is POSITIVE"))
+   } else if (x < 0) {
+     return(paste(x, "is NEGATIVE"))
+   } else {
+     return(paste(x, "is ZERO"))
+   }
+ }
>
> # Test with various numbers
> test_numbers <- c(5, -3, 0, 2.5, -7.8, 0.001, -0.001)
>
> cat("Testing number classification:\n")
Testing number classification:
> cat("-----\n")
-----
> for (num in test_numbers) {
+   result <- classify_number(num)
+   cat(result, "\n")
+ }
5 is POSITIVE
-3 is NEGATIVE
0 is ZERO
2.5 is POSITIVE
-7.8 is NEGATIVE
0.001 is POSITIVE
-0.001 is NEGATIVE
>
> # Extended version that can handle vectors
> classify_vector <- function(vec) {

```

```

+ result <- character(length(vec))
+ for (i in 1:length(vec)) {
+   if (vec[i] > 0) {
+     result[i] <- paste(vec[i], "is POSITIVE")
+   } else if (vec[i] < 0) {
+     result[i] <- paste(vec[i], "is NEGATIVE")
+   } else {
+     result[i] <- paste(vec[i], "is ZERO")
+   }
+ }
+ return(result)
+ }
>
> cat("\nClassifying entire vector:\n")

Classifying entire vector:
> cat("-----\n")
-----
> vector_results <- classify_vector(test_numbers)
> print(vector_results)
[1] "5 is POSITIVE"   "-3 is NEGATIVE"   "0 is ZERO"
[4] "2.5 is POSITIVE" "-7.8 is NEGATIVE" "0.001 is POSITIVE"
[7] "-0.001 is NEGATIVE"
>
> # Interactive version (uncomment to run interactively)
> # cat("\nInteractive mode (enter a number):\n")
> # user_input <- as.numeric(readline(prompt = "Enter a number: "))
> # cat(classify_number(user_input), "\n")
>
> cat("\n")

```

```

> #=====
> # ACTIVITY 2: Sum Numbers from 1 to 100 Using Loops
> #=====
>
> cat("=====\n")
=====
> cat("ACTIVITY 2: Summing Numbers from 1 to 100\n")
ACTIVITY 2: Summing Numbers from 1 to 100
> cat("=====\n\n")
=====
>

```

```

> # Method 1: Using a for loop
> cat("Method 1: Using for loop\n")
Method 1: Using for loop
> cat("-----\n")
-----
> sum_for <- 0
> for (i in 1:100) {
+   sum_for <- sum_for + i
+ }
> cat("Sum of 1 to 100 (for loop):", sum_for, "\n\n")
Sum of 1 to 100 (for loop): 5050

>
> # Method 2: Using a while loop
> cat("Method 2: Using while loop\n")
Method 2: Using while loop
> cat("-----\n")
-----
> sum_while <- 0
> counter <- 1
> while (counter <= 100) {
+   sum_while <- sum_while + counter
+   counter <- counter + 1
+ }
> cat("Sum of 1 to 100 (while loop):", sum_while, "\n\n")
Sum of 1 to 100 (while loop): 5050

>
> # Method 3: Using a repeat loop with break
> cat("Method 3: Using repeat loop\n")
Method 3: Using repeat loop
> cat("-----\n")
-----
> sum_repeat <- 0
> counter <- 1
> repeat {
+   sum_repeat <- sum_repeat + counter
+   counter <- counter + 1
+   if (counter > 100) {
+     break
+   }
+ }
> cat("Sum of 1 to 100 (repeat loop):", sum_repeat, "\n\n")
Sum of 1 to 100 (repeat loop): 5050

```

```

>
> # Method 4: Using vectorized approach (for comparison)
> cat("Method 4: Using vectorized approach\n")
Method 4: Using vectorized approach
> cat("-----\n")
-----
> sum_vectorized <- sum(1:100)
> cat("Sum of 1 to 100 (vectorized):", sum_vectorized, "\n\n")
Sum of 1 to 100 (vectorized): 5050

>
> # Verify all methods give the same result
> cat("Verification:\n")
Verification:
> cat("-----")
-----> cat("\nAll methods match:",
+   sum_for == sum_while && sum_while == sum_repeat && sum_repeat ==
sum_vectorized, "\n")

All methods match: TRUE
> cat("Expected sum (n*(n+1)/2):", 100*101/2, "\n")
Expected sum (n*(n+1)/2): 5050
>
> cat("\n")

```

```

=====
> cat("ACTIVITY 3: Random Number Generation Until > 0.9\n")
ACTIVITY 3: Random Number Generation Until > 0.9
> cat("===== \n\n")
=====

>
> # Method 1: Using repeat loop
> cat("Method 1: Using repeat loop\n")
Method 1: Using repeat loop
> cat("-----\n")
-----
> set.seed(123) # For reproducibility
> count_repeat <- 0
> repeat {
+   num <- runif(1) # Generate one random uniform number
+   count_repeat <- count_repeat + 1

```

```

+ cat("Iteration", count_repeat, ":", round(num, 4))
+ if (num > 0.9) {
+   cat(" *** TARGET EXCEEDED ***\n")
+   break
+ } else {
+   cat("\n")
+ }
+ }
Iteration 1 : 0.2876
Iteration 2 : 0.7883
Iteration 3 : 0.409
Iteration 4 : 0.883
Iteration 5 : 0.9405 *** TARGET EXCEEDED ***
> cat("Total iterations needed (repeat loop):", count_repeat, "\n\n")
Total iterations needed (repeat loop): 5

>
> # Method 2: Using while loop
> cat("Method 2: Using while loop\n")
Method 2: Using while loop
> cat("-----\n")
-----
> set.seed(123) # Reset seed for fair comparison
> count_while <- 0
> num <- 0
> while (num <= 0.9) {
+   num <- runif(1)
+   count_while <- count_while + 1
+   cat("Iteration", count_while, ":", round(num, 4))
+   if (num > 0.9) {
+     cat(" *** TARGET EXCEEDED ***\n")
+   } else {
+     cat("\n")
+   }
+ }
Iteration 1 : 0.2876
Iteration 2 : 0.7883
Iteration 3 : 0.409
Iteration 4 : 0.883
Iteration 5 : 0.9405 *** TARGET EXCEEDED ***
> cat("Total iterations needed (while loop):", count_while, "\n\n")
Total iterations needed (while loop): 5

>

```

```

> # Method 3: Generate until condition is met (simulated approach)
> cat("Method 3: Simulation of multiple trials\n")
Method 3: Simulation of multiple trials
> cat("-----\n")
-----
>
> # Function to run one trial and return number of iterations
> run_trial <- function(seed = NULL) {
+   if (!is.null(seed)) set.seed(seed)
+   count <- 0
+   repeat {
+     num <- runif(1)
+     count <- count + 1
+     if (num > 0.9) break
+   }
+   return(count)
+ }
>
> # Run multiple trials to see distribution
> trials <- 20
> results <- numeric(trials)
> cat("Running", trials, "trials:\n")
Running 20 trials:
> for (i in 1:trials) {
+   results[i] <- run_trial()
+   cat("Trial", i, ":", results[i], "iterations\n")
+ }
Trial 1 : 6 iterations
Trial 2 : 9 iterations
Trial 3 : 4 iterations
Trial 4 : 7 iterations
Trial 5 : 1 iterations
Trial 6 : 55 iterations
Trial 7 : 17 iterations
Trial 8 : 3 iterations
Trial 9 : 4 iterations
Trial 10 : 3 iterations
Trial 11 : 4 iterations
Trial 12 : 8 iterations
Trial 13 : 13 iterations
Trial 14 : 50 iterations
Trial 15 : 1 iterations
Trial 16 : 3 iterations
Trial 17 : 2 iterations

```

```

Trial 18 : 7 iterations
Trial 19 : 18 iterations
Trial 20 : 2 iterations
>
> cat("\nSummary of", trials, "trials:\n")

Summary of 20 trials:
> cat(" Minimum iterations:", min(results), "\n")
Minimum iterations: 1
> cat(" Maximum iterations:", max(results), "\n")
Maximum iterations: 55
> cat(" Mean iterations:", round(mean(results), 2), "\n")
Mean iterations: 10.85
> cat(" Median iterations:", median(results), "\n")
Median iterations: 5
> cat(" Theoretical mean (1/0.1 = 10): 10\n")
Theoretical mean (1/0.1 = 10): 10
>
> cat("\n")

```

```

> #=====
> # ACTIVITY 4: Rewrite a Loop Using Vectorisation
> #=====
>
> cat("===== \n")
=====
> cat("ACTIVITY 4: Loop vs Vectorisation Comparison\n")
ACTIVITY 4: Loop vs Vectorisation Comparison
> cat("===== \n\n")
=====

>
> # Create a large vector for testing
> test_vector <- 1:100000
>
> # EXAMPLE 1: Squaring numbers
> cat("EXAMPLE 1: Squaring numbers\n")
EXAMPLE 1: Squaring numbers
> cat("----- \n")
-----

>
> # Loop approach

```

```

> cat("Loop approach:\n")
Loop approach:
> start_time <- Sys.time()
> squared_loop <- numeric(length(test_vector))
> for (i in 1:length(test_vector)) {
+   squared_loop[i] <- test_vector[i]^2
+ }
> end_time <- Sys.time()
> loop_time <- difftime(end_time, start_time, units = "secs")
> cat(" Time taken:", round(loop_time, 4), "seconds\n")
  Time taken: 0.0768 seconds
> cat(" First 5 results:", squared_loop[1:5], "\n")
  First 5 results: 1 4 9 16 25
>
> # Vectorised approach
> cat("\nVectorised approach:\n")

Vectorised approach:
> start_time <- Sys.time()
> squared_vector <- test_vector^2
> end_time <- Sys.time()
> vector_time <- difftime(end_time, start_time, units = "secs")
> cat(" Time taken:", round(vector_time, 4), "seconds\n")
  Time taken: 0.0284 seconds
> cat(" First 5 results:", squared_vector[1:5], "\n")
  First 5 results: 1 4 9 16 25
> cat("\n Speed improvement:", round(loop_time/vector_time, 2), "x faster\n\n")
Error in `/.difftime`(loop_time, vector_time) :
  second argument of / cannot be a "difftime" object
>
> # EXAMPLE 2: Conditional operations (if x > 50000 then sqrt else x^2)
> cat("EXAMPLE 2: Conditional operations\n")
EXAMPLE 2: Conditional operations
> cat("-----\n")
-----
>
> # Loop approach
> cat("Loop approach:\n")
Loop approach:
> start_time <- Sys.time()
> result_loop <- numeric(length(test_vector))
> for (i in 1:length(test_vector)) {
+   if (test_vector[i] > 50000) {
+     result_loop[i] <- sqrt(test_vector[i])

```

```

+ } else {
+   result_loop[i] <- test_vector[i]^2
+ }
+ }
> end_time <- Sys.time()
> loop_time <- difftime(end_time, start_time, units = "secs")
> cat(" Time taken:", round(loop_time, 4), "seconds\n")
  Time taken: 0.1233 seconds
> cat(" First 10 results:", result_loop[1:10], "\n")
  First 10 results: 1 4 9 16 25 36 49 64 81 100
>
> # Vectorised approach using ifelse
> cat("\nVectorised approach (ifelse):\n")

Vectorised approach (ifelse):
> start_time <- Sys.time()
> result_vector <- ifelse(test_vector > 50000, sqrt(test_vector), test_vector^2)
> end_time <- Sys.time()
> vector_time <- difftime(end_time, start_time, units = "secs")
> cat(" Time taken:", round(vector_time, 4), "seconds\n")
  Time taken: 0.0452 seconds
> cat(" First 10 results:", result_vector[1:10], "\n")
  First 10 results: 1 4 9 16 25 36 49 64 81 100
> cat("\n Speed improvement:", round(loop_time/vector_time, 2), "x faster\n\n")
Error in `/.difftime`(loop_time, vector_time) :
  second argument of / cannot be a "difftime" object
>
> # EXAMPLE 3: Sum of squares
> cat("EXAMPLE 3: Sum of squares\n")
EXAMPLE 3: Sum of squares
> cat("-----\n")
-----
>
> # Loop approach
> cat("Loop approach:\n")
Loop approach:
> start_time <- Sys.time()
> sum_squares_loop <- 0
> for (i in 1:length(test_vector)) {
+   sum_squares_loop <- sum_squares_loop + test_vector[i]^2
+ }
> end_time <- Sys.time()
> loop_time <- difftime(end_time, start_time, units = "secs")
> cat(" Time taken:", round(loop_time, 4), "seconds\n")

```

```
Time taken: 0.0716 seconds
> cat(" Result:", sum_squares_loop, "\n")
Result: 3.333383e+14
```

Running the F-Scripts (F1 - F4)

```
> cat("=====\n")
=====
> cat("RUNNING SCRIPT F1: Basic Functions (triple function)\n")
RUNNING SCRIPT F1: Basic Functions (triple function)
> cat("=====\n\n")
=====

>
> # Source and run F-1-BasicFunctions.R
> # Make sure the file is in your working directory
> # You can either source it or copy the code here:
>
> cat("F1 - Triple Function Demo:\n")
F1 - Triple Function Demo:
> cat("-----\n")
-----

>
> # Function Definition
> triple <- function(x) {
+   x <- x * 3
+   x
+ }
>
> # Main Script
> a <- 5
> cat("a =", a, "\n")
a = 5
> cat("triple(a) =", triple(a), "\n")
triple(a) = 15
> cat("a (unchanged) =", a, "\n\n")
a (unchanged) = 5

>
> b <- c(2.1, 5.3, 7.9)
> cat("b =", paste(b, collapse = ", "), "\n")
b = 2.1, 5.3, 7.9
> cat("triple(b) =", paste(triple(b), collapse = ", "), "\n")
```

```

triple(b) = 6.3, 15.9, 23.7
> cat("b (unchanged) =", paste(b, collapse = ", "), "\n\n")
b (unchanged) = 2.1, 5.3, 7.9

>
> c <- c(TRUE, FALSE, TRUE)
> cat("c =", paste(c, collapse = ", "), "\n")
c = TRUE, FALSE, TRUE
> cat("triple(c) =", paste(triple(c), collapse = ", "),
+ "(TRUE=1, FALSE=0 in arithmetic)\n\n")
triple(c) = 3, 0, 3 (TRUE=1, FALSE=0 in arithmetic)

>
> d <- c("A", "B", "C")
> cat("d =", paste(d, collapse = ", "), "\n")
d = A, B, C
> cat("triple(d) = ")
triple(d) = > tryCatch({
+ print(triple(d))
+ }, error = function(e) {
+ cat("ERROR: Cannot multiply strings!\n")
+ })
ERROR: Cannot multiply strings!
>
> cat("\n")

> readline(prompt = "Press [Enter] to run F2...")
Press [Enter] to run F2...
[1] ""
> #=====
> cat("\n=====\\n")

=====
> cat("RUNNING SCRIPT F2: Exercise 1 - Three Ways to Calculate Mean\\n")
RUNNING SCRIPT F2: Exercise 1 - Three Ways to Calculate Mean
> cat("=====\\n\\n")

```

```

=====
> cat("RUNNING SCRIPT F2: Exercise 1 - Three Ways to Calculate Mean\\n")
RUNNING SCRIPT F2: Exercise 1 - Three Ways to Calculate Mean
> cat("=====\\n\\n")
=====

```

```

>
> # F-2-Exercise1.R
>
> # Function Definitions
> mean_mean <- function(x) {
+   out <- mean(x)
+   out
+ }
>
> mean_loop <- function(x) {
+   running_total <- 0
+   element_total <- 0
+   for (i in x) {
+     running_total <- running_total + i
+     element_total <- element_total + 1
+   }
+   out <- running_total / element_total
+   out
+ }
>
> mean_function <- function(x) {
+   out <- sum(x) / length(x)
+   out
+ }
>
> # Main script
> int_test <- 1:5
> float_test <- c(1.2, 2.5, 3.5, 4.5, 5.5)
>
> cat("Test Vectors:\n")
Test Vectors:
> cat("Integer vector:", paste(int_test, collapse = ", "), "\n")
Integer vector: 1, 2, 3, 4, 5
> cat("Float vector: ", paste(float_test, collapse = ", "), "\n\n")
Float vector: 1.2, 2.5, 3.5, 4.5, 5.5

>
> cat("Results using mean_mean (built-in mean):\n")
Results using mean_mean (built-in mean):
> cat(" Integer vector mean:", mean_mean(int_test), "\n")
Integer vector mean: 3
> cat(" Float vector mean: ", mean_mean(float_test), "\n\n")
Float vector mean: 3.44

```

```

>
> cat("Results using mean_loop (custom loop):\n")
Results using mean_loop (custom loop):
> cat(" Integer vector mean:", mean_loop(int_test), "\n")
Integer vector mean: 3
> cat(" Float vector mean: ", mean_loop(float_test), "\n\n")
Float vector mean: 3.44

>
> cat("Results using mean_function (sum/length):\n")
Results using mean_function (sum/length):
> cat(" Integer vector mean:", mean_function(int_test), "\n")
Integer vector mean: 3
> cat(" Float vector mean: ", mean_function(float_test), "\n\n")
Float vector mean: 3.44

>
> cat("All methods produce the same results!\n")
All methods produce the same results!
>
> cat("\n")

```

```

=====
> cat("RUNNING SCRIPT F3: Multiple Parameters\n")
RUNNING SCRIPT F3: Multiple Parameters
> cat("===== \n\n")
=====

>
> # F-3-MultipleParameters.R
>
> # Function Definitions
> raise <- function(x, y) x^y
>
> echostring <- function(n, string) {
+   return_vector <- NA
+   for (i in 1:length(string)) {
+     for (j in 1:n) {
+       return_vector <- c(return_vector, string[i])
+     }
+   }
+   return(return_vector[-1])
+ }

```

```

>
> # Main Script
> cat("raise(x,y) function - raise x to power y:\n")
raise(x,y) function - raise x to power y:
> cat("-----\n")
-----
> cat("raise(2,3) =", raise(2, 3), "\n")
raise(2,3) = 8
>
> a <- c(2, 3)
> cat("a = c(2,3)\n")
a = c(2,3)
> cat("raise(a[1], a[2]) =", raise(a[1], a[2]), "\n")
raise(a[1], a[2]) = 8
>
> b <- c(3, 4)
> cat("b = c(3,4)\n")
b = c(3,4)
> cat("raise(a, b) =", paste(raise(a, b), collapse = " "),
+ "(element-wise operation)\n\n")
raise(a, b) = 8, 81 (element-wise operation)

>
> cat("echostring(n, string) function - repeat strings:\n")
echostring(n, string) function - repeat strings:
> cat("-----\n")
-----
> outstring <- echostring(3, "Echo...")
> cat("echostring(3, 'Echo...') =", paste(outstring, collapse = " "), "\n")
echostring(3, 'Echo...') = Echo..., Echo..., Echo...
>
> c <- c("A string", "Another string")
> outstring <- echostring(3, c)
> cat("echostring(3, c('A string', 'Another string')) =\n")
echostring(3, c('A string', 'Another string')) =
> cat(" ", paste(outstring, collapse = " "), "\n")
  A string, A string, A string, Another string, Another string, Another string
>
> cat("\n")

```

```

=====
> cat("RUNNING SCRIPT F4: Exercise 2 - Building Functions\n")
RUNNING SCRIPT F4: Exercise 2 - Building Functions

```

```

> cat("=====\\n\\n")
=====

>
> # F-4-Exercise2.R
>
> # Function Definitions
> build_range <- function(x, y, size) {
+   outvector <- seq(x, y, by = size)
+   return(outvector)
+ }
>
> print_squares <- function(v, limit) {
+   for (i in v) {
+     if (i^2 <= limit) {
+       print(i^2)
+     }
+   }
+ }
>
> print_limit <- function(x, y, size, limit) {
+   numberlist <- build_range(x, y, size)
+   print_squares(numberlist, limit)
+ }
>
> # Main Script
> cat("Test build_range(4,13,4):\\n")
Test build_range(4,13,4):
> cat(" ", paste(build_range(4, 13, 4), collapse = ", "), "\\n\\n")
  4, 8, 12

>
> cat("Test print_squares(3:9,40):\\n")
Test print_squares(3:9,40):
> cat(" Squares <= 40 from numbers 3 to 9:\\n")
  Squares <= 40 from numbers 3 to 9:
> print_squares(3:9, 40)
[1] 9
[1] 16
[1] 25
[1] 36
> cat("\\n")

>

```

```
> cat("Test print_limit(2,9,3,30):\n")
Test print_limit(2,9,3,30):
> cat(" For numbers from 2 to 9 by 3, print squares <= 30:\n")
  For numbers from 2 to 9 by 3, print squares <= 30:
> print_limit(2, 9, 3, 30)
[1] 4
[1] 25
> cat("\n")

>
> cat("Test with floats - print_limit(2.1,15.6,2.5,35.2):\n")
Test with floats - print_limit(2.1,15.6,2.5,35.2):
> cat(" For numbers from 2.1 to 15.6 by 2.5, print squares <= 35.2:\n")
  For numbers from 2.1 to 15.6 by 2.5, print squares <= 35.2:
> print_limit(2.1, 15.6, 2.5, 35.2)
[1] 4.41
[1] 21.16
> cat("\n")
```