R Programming Basics

Armand Tossou

9/5/2021

Logical Operators

- & # AND
- # OR
- ! # NOT

```
x <- 10

(x>5) & (x < 20) # TRUE
```

```
## [1] TRUE
```

```
(10 == 1) # FALSE
```

```
## [1] FALSE
```

```
!(10 == 1) # TRUE
```

```
## [1] TRUE
```

```
## a more realistic example
df <- mtcars
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
# what cars have miles per galon greater than 20?
df[df$mpg > 20, ]
```

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona  21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

```
subset(df, mpg > 20) # alternatively
```

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona  21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

```
# using 2 conditions
df[(df$mpg >20) & (df$hp > 100),]
```

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

If, else, and else if Statements

```
# assign value to x
x <- 13

# create conditional statements
if (x==10) {
  # code
  print('x is equal to 10!')
} else if (x==12) {
  print('x is equal to 12')
} else {
  print('x was not equal to 10 or 12')
}
```

```
## [1] "x was not equal to 10 or 12"
```

Another example:

```
temp <- 75

if (temp > 80) {# Execute if condition was TRUE
  print('Hot outside!')
} else if (temp < 80 & temp >= 50) {
    print('Nice outside!')
} else {
    print('It\'s less than 50 degrees outside')
  }
```

```
## [1] "Nice outside!"
```

Logical Operators with Vectors

We have two options when use logical operators, a comparison of the entire vectors element by element, or just a comparison of the first elements in the vectors, to make sure the output is a single Logical.

```
tf <- c(TRUE,FALSE)
tt <- c(TRUE,TRUE)
ft <- c(FALSE, TRUE)

# create the condition
tt & tf # TRUE FALSE
```

```
## [1]  TRUE FALSE
```

```
tt | tf # TRUE TRUE
```

```
## [1] TRUE TRUE
```

To compare first elements use && or ||.

```
ft && tt # FALSE
```

```
## [1] FALSE
```

```r
tt && tf # TRUE
```

```
## [1] TRUE
```

```r
tt || tf # TRUE
```

```
## [1] TRUE
```

```r
tt || ft # TRUE
```

```
## [1] TRUE
```

Conditional Statements Exercises

Example: Write a script that prints "Hello" if the variable x is equal to 1:

```r
x <- 1 # Change x to test

if (x==1) {
  print('Hello')
}
```

```
## [1] "Hello"
```

Ex 1: Write a script that will print "Even Number" if the variable x is an even number, otherwise print "Not Even":

```r
x <- 3 # Change x to test

if (x %% 2 == 0) {
  print("Even number")
} else {
  print("Not even")
}
```

```
## [1] "Not even"
```

Ex 2: Write a script that will print 'Is a Matrix' if the variable x is a matrix, otherwise print "Not a Matrix". Hint: You may want to check out help(is.matrix)

```r
x <- matrix()

if (is.matrix(x)) {
  print("Is a Matrix")
} else {
  print("Not a Matrix")
}
```

```
## [1] "Is a Matrix"
```

Ex 3: Create a script that given a numeric vector x with a length 3, will print out the elements in order from high to low. You must use if,else if, and else statements for your logic. (This code will be relatively long)

```r
#x <- c(3,7,1) # change order to test
#x <- c(1,3,7) # change order to test
x <- c(1,7,3) # change order to test

# print highest number
if ( x[1] == max(x) ) {
  print(x[1])
} else if ( x[2] == max(x) ) {
  print(x[2])
} else {
  print(x[3])
}
```

```
## [1] 7
```

```r
# print middle number
if ( (x[1] > x[2] & x[1] < x[3]) | (x[1] > x[3] & x[1] < x[2]) ) {
  print(x[1])
} else if ( (x[2] > x[1] & x[2] < x[3]) | (x[2] > x[3] & x[2] < x[1]) ) {
  print(x[2])
} else {
  print(x[3])
}
```

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

```r
# print lowest number
if ( (x[1] == min(x)) ) {
  print(x[1])
} else if ( (x[2] == min(x)) ) {
  print(x[2])
} else {
  print(x[3])
}
```

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

Ex 4: Write a script that uses if,else if, and else statements to print the max element in a numeric vector with 3 elements.

```r
x <- c(20, 10, 1) # change order to test

# print highest number
if ( x[1] == max(x) ) {
  print(x[1])
} else if ( x[2] == max(x) ) {
  print(x[2])
} else {
  print(x[3])
}
```

```
## [1] 20
```

## While Loops

```
x <- 0

while (x<10) {
  print( paste0('x is: ' , x) )
  x <- x+1
  if (x==10) {
    print("x is now equal to 10! Break loop!")
    break
    print("Wooo I printed too!")
  }
}
```

```
## [1] "x is: 0"
## [1] "x is: 1"
## [1] "x is: 2"
## [1] "x is: 3"
## [1] "x is: 4"
## [1] "x is: 5"
## [1] "x is: 6"
## [1] "x is: 7"
## [1] "x is: 8"
## [1] "x is: 9"
## [1] "x is now equal to 10! Break loop!"
```

```
x <- 0

while (x<10) {
  print( paste0('x is: ' , x) )
  x <- x+1
  if (x==5) {
    print("x is now equal to 5! Break loop!")
    break
  }
}
```

```
## [1] "x is: 0"
## [1] "x is: 1"
## [1] "x is: 2"
## [1] "x is: 3"
## [1] "x is: 4"
## [1] "x is now equal to 5! Break loop!"
```

For Loops

A ‘for loop’ allows us to iterate over objects.

```r
v <- c(1,2,3)

for (variable in v) {
  print(variable)
}
```

```
## [1] 1
## [1] 2
## [1] 3
```

```r
v <- c(1,2,3,4,5)

for (temp.var in v) {
  # Execute some code
  # for every temp.var in v
  print('Hello')
}
```

```
## [1] "Hello"
## [1] "Hello"
## [1] "Hello"
## [1] "Hello"
## [1] "Hello"
```

```r
v <- c(1,2,3,4,5)

for (temp.var in v) {
  result <- temp.var + 1
  print( paste0('The temp.var plus 1 is equal to:', result) )
  print('----------------------------------')
}
```

```
## [1] "The temp.var plus 1 is equal to:2"
## [1] "----------------------------------"
## [1] "The temp.var plus 1 is equal to:3"
## [1] "----------------------------------"
## [1] "The temp.var plus 1 is equal to:4"
## [1] "----------------------------------"
## [1] "The temp.var plus 1 is equal to:5"
## [1] "----------------------------------"
## [1] "The temp.var plus 1 is equal to:6"
## [1] "----------------------------------"
```

We can use for loops for other objects as well. Here’s an example to illustrate the case of a list

```r
my.list <- list(c(1,2,3), mtcars)

for (item in my.list) {
  print(item)
}
```

```
## [1] 1 2 3
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

Using a for loop with a matrix:

```r
mat <- matrix(1:25, byrow = TRUE, nrow = 5)

for (num in mat) {
  print(num)
}
```

```
## [1] 1
## [1] 6
## [1] 11
## [1] 16
## [1] 21
## [1] 2
## [1] 7
## [1] 12
## [1] 17
## [1] 22
## [1] 3
## [1] 8
## [1] 13
## [1] 18
## [1] 23
## [1] 4
## [1] 9
## [1] 14
## [1] 19
## [1] 24
## [1] 5
## [1] 10
## [1] 15
## [1] 20
## [1] 25
```

Nested for loops:

```
mat <- matrix(1:25, byrow = FALSE, nrow = 5)
mat
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    6   11   16   21
## [2,]    2    7   12   17   22
## [3,]    3    8   13   18   23
## [4,]    4    9   14   19   24
## [5,]    5   10   15   20   25
```

```
print("")
```

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

```
for (row in 1:nrow(mat)) {
  for (col in 1:ncol(mat)) {
    print( paste('The element at row:', row, 'and col:', col, 'is', mat[row,col]))
  }
}
```

```
## [1] "The element at row: 1 and col: 1 is 1"
## [1] "The element at row: 1 and col: 2 is 6"
## [1] "The element at row: 1 and col: 3 is 11"
## [1] "The element at row: 1 and col: 4 is 16"
## [1] "The element at row: 1 and col: 5 is 21"
## [1] "The element at row: 2 and col: 1 is 2"
## [1] "The element at row: 2 and col: 2 is 7"
## [1] "The element at row: 2 and col: 3 is 12"
## [1] "The element at row: 2 and col: 4 is 17"
## [1] "The element at row: 2 and col: 5 is 22"
## [1] "The element at row: 3 and col: 1 is 3"
## [1] "The element at row: 3 and col: 2 is 8"
## [1] "The element at row: 3 and col: 3 is 13"
## [1] "The element at row: 3 and col: 4 is 18"
## [1] "The element at row: 3 and col: 5 is 23"
## [1] "The element at row: 4 and col: 1 is 4"
## [1] "The element at row: 4 and col: 2 is 9"
## [1] "The element at row: 4 and col: 3 is 14"
## [1] "The element at row: 4 and col: 4 is 19"
## [1] "The element at row: 4 and col: 5 is 24"
## [1] "The element at row: 5 and col: 1 is 5"
## [1] "The element at row: 5 and col: 2 is 10"
## [1] "The element at row: 5 and col: 3 is 15"
## [1] "The element at row: 5 and col: 4 is 20"
## [1] "The element at row: 5 and col: 5 is 25"
```

## Functions

A function is just a useful device that groups together a set of statements so they can be run more than once.

```
# syntax

name_of_func <- function(input1,input2,input3=45) {
  # code to execute
  result <- input1 + input2 * input3
  return(result)
}
```

Built-in functions:

```
sum(c(1,2,3))
```

```
## [1] 6
```

```
help(sum)
```

```
## starting httpd help server ... done
```

Create a simple function that takes no input:

```
hello <- function() {
  # code that executes when the function is called
  print("Hello")
}

hello() # call the function
```

```
## [1] "Hello"
```

Create a function that takes an input:

```r
hello <- function(name) {
  print( paste("Hello", name))
}

hello('Sammy') # call the function
```

```
## [1] "Hello Sammy"
```

Assign a default value to the input:

```r
hello <- function(name='Frank') {
  print( paste("Hello", name))
}

hello() # call the function
```

```
## [1] "Hello Frank"
```

```r
hello('Sammy') # call the function
```

```
## [1] "Hello Sammy"
```

Create a function with multiple inputs:

```r
add_num <- function(num1,num2) {
  print(num1+num2)
}

add_num(4,5)
```

```
## [1] 9
```

Examples showing how we can return a function output:

```r
add_num <- function(num1,num2) {
  my.sum <- num1+num2
  return(my.sum)
}

result <- add_num(4,5)
result
```

```
## [1] 9
```

Variable scope:

```r
times5 <- function(num){
  return(num*5)
}

print(times5(20))
```

```
## [1] 100
```

Same function, with some additions:

```
times5 <- function(num){
  my.result <- num*5
  return(my.result)
}

my.output <- times5(100)

print(my.output)
```

```
## [1] 500
```

```
#my.result # we get this error message 'Error: object 'my.result' not found' because the variable is lo
```

Showing an example of a global vs local variable:

```
v <- "I'm a global variable"
stuff <- "I'm a global stuff"

fun <- function(stuff){
  print(v)
  stuff <- "Reassign stuff inside of this function fun"
  print(stuff)
}

fun(stuff) # call the function
```

```
## [1] "I'm a global variable"
## [1] "Reassign stuff inside of this function fun"
```

```
print(stuff)
```

```
## [1] "I'm a global stuff"
```

## Function Exercise

EXAMPLE 1: Create a function that takes in a name as a string argument, and prints out "Hello name"

```
hello <- function(name){
  print(paste('Hello', name))
}

hello('Armand')
```

```
## [1] "Hello Armand"
```

EXAMPLE 2: Create a function that takes in a name as a string argument and returns a string of the form - "Hello name"

```r
hello_2 <- function(name){
  result <- paste('Hello', name, sep=" ")
}

print(hello_2('Armand'))
```

```
## [1] "Hello Armand"
```

Ex 1: Create a function that will return the product of two integers.

```r
prod <- function(num1,num2) {
  return(num1 * num2)
}

prod(3,4)
```

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

Ex 2: Create a function that accepts two arguments, an integer and a vector of integers. It returns TRUE if the integer is present in the vector, otherwise it returns FALSE. Make sure you pay careful attention to your placement of the return(FALSE) line in your function!

```r
num_check <- function(int,vec) {
  if (int %in% vec) {
    return(TRUE)
  } else {
    return(FALSE)
  }
}

#num_check(2,c(1,2,3)) # change to test
num_check(7,c(1,2,3)) # change to test
```

```
## [1] FALSE
```

Ex 3: Create a function that accepts two arguments, an integer and a vector of integers. It returns the count of the number of occurrences of the integer in the input vector.

```r
num_count <- function(num,vec){
  # create a counter
  myCounter = 0
  for(i in vec){
    if(num == i){
      myCounter = myCounter+1
    }
  }
  return(myCounter)
}

#num_count(2,c(1,1,2,2,3,3)) # test numero 1. Answer: 2
num_count(1,c(1,1,2,2,3,1,4,5,5,2,2,1,3)) # test numero 2. Answer: 4
```

```
## [1] 4
```

Ex 4: We want to ship bars of aluminum. We will create a function that accepts an integer representing the requested kilograms of aluminum for the package to be shipped. To fullfill these orders, we have small bars (1 kilogram each) and big bars (5 kilograms each). Return the least number of bars needed.

For example, a load of 6 kg requires a minimum of two bars (1 5kg bars and 1 1kg bars). A load of 17 kg requires a minimum of 5 bars (3 5kg bars and 2 1kg bars).

```
minNumBars <- function(num){
  return( (num %/% 5) + (num %% 5) )
}

minNumBars(17) # Answer: 5
```

```
## [1] 5
```

```
minNumBars(6) # Answer: 2
```

```
## [1] 2
```

Ex 5: Create a function that accepts 3 integer values and returns their sum. However, if an integer value is evenly divisible by 3, then it does not count towards the sum. Return zero if all numbers are evenly divisible by 3. Hint: You may want to use the append() function.

```
summer <- function(int1,int2,int3){
  # create empty list
  ans <- 0
  # create conditions
  for (i in c(int1,int2,int3)){
    if (!(i %% 3==0)) {
      ans <- ans + i
    }
  }
  return(ans)
}

#summer(7,2,3) # Answer: 9
#summer(3,6,9) # Answer: 0
summer(9,11,12) # Answer: 11
```

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

Alternative solution:

```
summer <- function(a,b,c){
  # create output list
  out <- c(0)
  # create conditions
  if ( a %% 3 != 0){
    out <- append(out,a)
  }
  if ( b %% 3 != 0){
    out <- append(out,b)
  }
  if ( c %% 3 != 0){
    out <- append(out,c)
  }
  return(sum(out))
}

#summer(7,2,3) # Answer: 9
#summer(3,6,9) # Answer: 0
summer(9,11,12) # Answer: 11
```

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

Ex 6: Create a function that will return TRUE if an input integer is prime. Otherwise, return FALSE. You may want to look into the any() function. There are many possible solutions to this problem.

```
prime_check <- function(num){
  if (num<=1){
    return(FALSE)
  }
  if (num==2){
    return(TRUE)
  }
  for (x in 2:(num-1)){
    if (num %% x ==0){
      return(FALSE)
    }
  }
  return(TRUE)
}

prime_check(-30)
```

```
## [1] FALSE
```