Introductory Programming in R

By Asef Nazari

asef.nazari@monash.edu

Faculty of IT

Monash university

4. Controlling the Execution flow

4.1 Logical Expressions

R allows us to create logical expressions and vectors in order to manipulate logical quantities. To create logical vectors, you may use boolean vales TRUE, FALSE, or NA (for missing / not available) directly, in addition to using the condition/logic operations. Pay attention that R treats TRUE as 1 and FALSE as 0.

R Relational Operators

- $<$
- $>$
- $<=$
- $>=$
- ==
- $! =$

R Logical Operators

- $x\&y$ for (x and y): Element-wise logical AND
- $x\&\&y$: Logical AND
- $x|y$ for (x or y) Element-wise logical OR
- $x||y$: Logical OR
- $!x$ for (not x): Logical NOT

Operators $\&$ and | perform element-wise operation producing result having length of the longer operand. But $\&\&$ and || examines only the first element of the operands resulting into a single length logical vector.

```
In [2]:
```

```
2 > 3
```

FALSE

```
In [3]:
```

```
4 != 5
```

TRUE

In [4]:

```
(3 != 12) & (2.7 >= 1.9)
```

```
TRUE
```

In [5]:

```
y <- c(TRUE, TRUE, FALSE, TRUE, 5 > 2)
y
sum(y)
```

```
    TRUE  TRUE  FALSE  TRUE  TRUE

4
```

In [6]:

```
z <- 3
z>= 3 && z<7
z<10 || z>5
```

```
TRUE

TRUE
```

In [7]:

```
x <- c(1:10)
x[(x>8) | (x<5)]
```

```
    1  2  3  4  9  10
```

In [8]:

```
x <- c(1:10)
x[(x>=8) & (x>=5)]
```

```
    8  9  10
```

In [9]:

```
x <- 1
y <- 3
(x==1) & (y==3)
```

```
TRUE
```

In [10]:

```
(x==1) | (y!=3)
```

```
TRUE
```

In [11]:

```
x <- c(TRUE, TRUE, FALSE, TRUE, FALSE,  0, 5) # zero is considered FALSE, and nozer
y <- c(FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE)
```

In [12]:

```
x
```

```
1   1   0   1   0   0   5
```

In [13]:

```
!x
```

```
FALSE   FALSE   TRUE   FALSE   TRUE   TRUE   FALSE
```

In [14]:

```
x & y
```

```
FALSE   TRUE   FALSE   TRUE   FALSE   FALSE   FALSE
```

In [15]:

```
x && y
```

```
FALSE
```

In [16]:

```
x | y
```

```
TRUE   TRUE   FALSE   TRUE   FALSE   TRUE   TRUE
```

In [17]:

```
x || y
```

```
TRUE
```

In [18]:

```
x <- c(TRUE, TRUE, FALSE, TRUE, FALSE)
y <- c(TRUE, TRUE,TRUE, TRUE,TRUE)

x && y
```

```
TRUE
```

In [19]:

```
#ifelse function
x <- seq(10)
ifelse(x %% 2 == 0,"even","odd")
```

```
'odd'   'even'   'odd'   'even'   'odd'   'even'   'odd'   'even'   'odd'   'even'
```

4.2 Control Structures

Helps you to control the flow of execution of the program

- if, else: to check a condition
- for: to loop for a fixed number of times

- while: to loop while a condition is TRUE
- break: to break a loop
- next: to skip an iteration
- return: to exit a function

If structure

```
#### if statement
if (test_expression) {
   statement
}

#### if-else

if (test_expression) {
   statement1
} else {
   statement2
}

#### Nested if

if ( test_expression1) {
   statement1
} else if ( test_expression2) {
   statement2
} else if ( test_expression3) {
   statement3
} else
   statement4
```

In [20]:

```
x <- 2
if(x == 2){
    print("Yesss")
}
```

```
[1] "Yesss"
```

In [21]:

```
if(x > 2){
    print("Greater")
} else if(x < 2) {
    print("Smaller")
} else {
    print("Equal")
}
```

```
[1] "Equal"
```

any() and all() functions

In [22]:

```
x <- 1:10
if (any(x > 4)) print("Well done!")
if (any(x > 12)) print("No Way!")
if (all(x > 7)) print("Another one!")
if (all(x > 0)) print("Hit the road!")
```

```
[1] "Well done!"
[1] "Hit the road!"
```

For structure

```
for (val in sequence)
{
    statement
}
```

In [23]:

```
for (i in 1:5){
    print(i)
}
```

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

In [24]:

```
y <- c("a", "b", "c", "d")
# makes loops iterations based on length of y
for (i in seq_along(y)){
    print(i)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
```

In [25]:

```
seq_along(6)
```

1

In [26]:

```
for(k in c("a", "b", "c", "d")){
    print(k)
}
```

```
[1] "a"
[1] "b"
[1] "c"
[1] "d"
```

In [27]:

```
# nested for loop
m <- matrix(nrow=2, ncol=3)
for (i in 1:nrow(m)){
    for(j in 1:ncol(m)){
        m[i,j] <- i*j
    }
}
m
```

1	2	3
2	4	6

In [28]:

```
count <- 5
while(count >0){
    print(count)
    count <- count -1
}
```

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

In [29]:

```
for(i in 1:10){
    if(i %% 2==0){
        next
    }
    print(i)
}
```

```
[1] 1
[1] 3
[1] 5
[1] 7
[1] 9
```