R PROGRAMMING FOR DATA SCIENCE

CENTRE FOR DATA SCIENCE AND ANALYTICS (C4DSA)

Session: 3/10 – Control Structures (Pre)
Trainer : Dr. Thulasyammal Ramiah Pillai
Date: 4th July, 2018

http://c4dsa.taylors.edu.my

Introduction to the R Language

Control Structures

Control Structures

Control structures in R allow you to control the flow of execution of the program, depending on runtime conditions. Common structures are

- `if`, `else`: testing a condition
- `for`: execute a loop a fixed number of times
- `while`: execute a loop *while* a condition is true
- `repeat`: execute an infinite loop
- `break`: break the execution of a loop
- `next`: skip an interation of a loop
- `return`: exit a function

Most control structures are not used in interactive sessions, but rather when writing functions or longer expresisons.

if, else

METHOD 1

```
if(<condition>) {
## do something
} else {
## do something else
}
```

METHOD 2

```
if(<condition1>) {
## do something
} else if(<condition2>) {
## do something different
} else {
## do something different
}
```

METHOD 3

```
if(<condition1>) {
}
if(<condition2>) {
}
```

if, else

EXAMPLE 1

```
if(x > 3) {
y <- 10
} else {
y <- 0
}
```

EXAMPLE 2

```
y <- if(x > 3) {
10
} else {
0
}
```

EXAMPLE 3

```
y <- if(0<x <5 {
10
}
If(6<x<10) {
0
}
```

for

`for` loops take an interator variable and assign it successive values from a sequence or vector. For loops are most commonly used for iterating over the elements of an object (list, vector, etc.)

```
for(i in 1:10) {
        print(i)
}
```

This loop takes the `i` variable and in each iteration of the loop gives it values 1, 2, 3, ..., 10, and then exits.

for

These three loops have the same behavior.

```
x <- c("a", "b", "c","d")

for(i in 1:4) {
        print(x[i])
}

for(i in seq_along(x)) {
        print(x[i])
}

for(letter in x) {
        print(letter)
}

for(i in 1:4) print(x[i])
```

Nested forloops

`for` loops can be nested.

```
x <- matrix(1:6, 2, 3)

for(i in seq_len(nrow(x))) {
        for(j in seq_len(ncol(x))) {
                print(x[i, j])
        }
}
```

Be careful with nesting though. Nesting beyond 2–3 levels is often very difficult to read/understand.

while

While loops begin by testing a condition. If it is true, then they execute the loop body. Once the loop body is executed, the condition is tested again, and so forth.

```
count <- 0
while(count < 10) {
        print(count)
        count <- count + 1
}
```

While loops can potentially result in infinite loops if not written properly. Use with care!

while

Sometimes there will be more than one condition in the test.

```
z <- 5

while(z >= 3 && z <= 10) {
        print(z)
        coin <- rbinom(1, 1, 0.5)

        if(coin == 1) {  ## random walk
                z <- z + 1
        } else {
                z <- z - 1
        }
}
```

Conditions are always evaluated from left to right.

repeat

Repeat initiates an infinite loop; these are not commonly used in statistical applications but they do have their uses. The only way to exit a `repeat` loop is to call `break`.

```
x <- 1
repeat {
print(x)
x = x+1
if (x == 6){
break
}
}
```

```
for(i in 1:100){
        print(i)
        if(i>20){
            break
        }
    }
```

next, return

`next` is used to skip an iteration of a loop

```
for(i in 1:100) {
        if(i <= 20) {
                ## Skip the first 20 iterations
                next
        }
        ## Do something here
}
```

`return` signals that a function should exit and return a given value

```
for (i in 1:10) {
        if (i < 5) next
        print(i)
        }
```

```
x <- 1:100

for(i in 1:10) {
        if(x[i] < 5) next
        print(x[i])
        }
```

```
check <- function(x) {
        if (x>0) {
                return("Positive")
                }
        else if (x<0) {
                return("Negative")
                }
        else {
                return("Zero")
                }
        }
```

Control Structures

Summary

- Control structures like `if`, `while`, and `for` allow you to control the flow of an R program
- Infinite loops should generally be avoided, even if they are theoretically correct.
- Control structures mentiond here are primarily useful for writing programs; for command-line interactive work, the *apply functions are more useful.