Control Structures

Dr.Sc. Oleksii Yehorchenkov

Department of Spatial Planning

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 iteration of a loop
- `return`: exit a function

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

Control Structures: if

```
 1  if(<condition>) {
 2      ## do something
 3  } else {
 4      ## do something else
 5  }
 6
 7  if(<condition1>) {
 8      ## do something
 9  } else if(<condition2>)  {
10      ## do something different
11  } else {
12      ## do something different
13  }
```

if

This is a valid `if/else` structure.

```
1  if(x > 3) {
2      y <- 10
3  } else {
4      y <- 0
5  }
```

So is this one.

```
1  y <- if(x > 3) {
2      10
3  } else {
4      0
5  }
```

if

Of course, the else clause is not necessary.

```
1  if(<condition1>) {
2  }
3
4  if(<condition2>) {
5  }
```

for

for loops take an iterator 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.)

```
1  for(i in 1:5) {
2      print(i)
3  }
```

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

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.

```
1
2  x <- c("a", "b", "c", "d")
3
4  for(i in 1:4) {
5      print(x[i])
6  }
7
8  for(i in seq_along(x)) {
9      print(x[i])
10 }
11
12 for(letter in x) {
13     print(letter)
14 }
15
16 for(i in 1:4) print(x[i])
```

Nested **for** loops

for loops can be nested.

```
1  x <- matrix(1:6, 2, 3)
2
3  for(i in seq_len(nrow(x))) {
4      for(j in seq_len(ncol(x))) {
5          print(x[i, j])
6      }
7  }
```

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.

```
1  count <- 0
2  while(count < 10) {
3      print(count)
4      count <- count + 1
5  }
```

`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.

```
 1  z <- 5
 2  while(z >= 3 && z <= 10) {
 3      print(z)
 4      coin <- rbinom(1, 1, 0.5)
 5          if(coin == 1) {   ## random walk
 6              z <- z + 1
 7          } else {
 8              z <- z - 1
 9          }
10  }
```

Conditions are always evaluated from left to right.

repeate

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.

```
 1  x0 <- 1
 2  tol <- 1e-8
 3
 4  repeat {
 5      x1 <- computeEstimate()
 6
 7      if(abs(x1 - x0) < tol) {
 8          break
 9      } else {
10          x0 <- x1
11      }
12  }
```

repeate loop a bit dangerous because there's no guarantee it will stop. Better to set a hard limit on the number of iterations (e.g. using a for loop) and then report whether convergence was achieved or not.

next, return

next is used to skip an iteration of a loop

```
1  for(i in 1:100) {
2      if (i <= 20) {
3          ## Skip the first 20 iterations
4          next
5      }
6      ## Do something here
7  }
```

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

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 mentioned here are primarily useful for writing programs; for command-line interactive work, the `*apply` functions are more useful.