Module 2

Nathan Cardoso

Control structures in R

Control structures in R allow you to control the flow of execution of teh program, depending on the runtime conditions. Some common structures are if-else. for, while, repeat, next, break and return.

The if control structure

The general syntax is as follows. The else section is optional. Suppose you want to execute something based on a certain condition but do nothing if the condition is not true; it is possible to use just an if.

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

We can also have multiple if conditiions as follows

```
if (<condition>) {
  ## do something
} else if (<condition2>){
  ## do something diff
} else {
  ## do something diff
}
```

This is a simple example of an if-else.

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

```
## [1] 10
```

Another way of doing the same is:

```r
x <- 7
y <- if(x>3) {
  10
} else {
  0
}
y
```

```
## [1] 10
```

The for loop

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 elements of an object (list, vector, etc).

```r
for(i in 1:10){
  print(2*i)
}
```

```
## [1] 2
## [1] 4
## [1] 6
## [1] 8
## [1] 10
## [1] 12
## [1] 14
## [1] 16
## [1] 18
## [1] 20
```

This loop takes the i variable and in each iteration of the loop assigns it a value starting from 1. It exits when the vector 1:10 comes to an end.

There are different ways to use a for loop when we want to index the elements in various R objects. All the three loops written above have the same behavior.

```r
x <- c("a","b","c","d")
for(i in 1:4) {
  print(x[i])
}
```

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

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

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

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

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

We can also have nested for loops. A use-case could be iterating through the elements of a matrix.

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

```
## [1] 1
## [1] 3
## [1] 5
## [1] 2
## [1] 4
## [1] 6
```

However, nesting is not advisable beyond 3-4 levels as it becomes difficult to read and understand.

The while loop

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.

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

```
## [1] 0
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
```

While loops can potentially result in infinite loops if not written properly. You have to make sure that the condition that stops the loop will actually occur. It is considered safer to use a for loop that has a definite limit on the number of times it will execute.

We can also test multiple conditions in a while loop

```
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
    }
}
```

```
## [1] 5
## [1] 4
## [1] 3
```

Conditions are always evaluated from left to right.

repeat, next and break

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

```
x0 <- 1
tol <- 1e-8

repeat {
    x1 <- computeEstimate()
    if(abs(x1-x0) < tol){
      break
    } else {
      x0 <- x1
  }
}
```

The loop above is a bit dangerous because there is no guarantee it will stop. It depends entirely on the result of the computeEstimate() function. It is better to set a hard limit on the number of iterations (e.g. using a for loop) and then report whether convergence between x0 and x1 was achieved or not.

next is used to skip an iteration of a loop

```
for(i in 1:100){
  if(i<=10){
    ## Skip the first 20 iterations
    next
  }
  ## Do something here
}
```

The return signals that a function should exit and return a given value. It is sometimes used with loops as well.

Functions