Control structures

#Here is an example of a valid if/else structure.

#The runif() function generates random deviates of the uniform distribution and is written as runif(n, min = 0, max = 1)

```
> x <- runif(1, 0, 10)
> if(x > 3) {
+   y <- 10
+ } else {
+   y <- 0
+ }
> y
[1] 10

>
```

The value of y is set depending on whether x > 3 or not. This expression can also be written a different, but equivalent, way in R.

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

In R, 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)
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

```
> x <- c("a", "b", "c", "d")
>
> for(i in 1:4) {
+       ## Print out each element of 'x'
+       print(x[i])
+ }
[1] "a"
[1] "b"
[1] "c"
[1] "d"
```

The seq_along() function is commonly used in conjunction with for loops in order to generate an integer sequence based on the length of an object (in this case, the object x).

```
> ## Generate a sequence based on length of 'x'
> for(i in seq_along(x)) {
+       print(x[i])
+ }
[1] "a"
[1] "b"
[1] "c"
[1] "d"
```

For one line loops, the curly braces are not strictly necessary.

```
> for(i in 1:4) print(x[i])
[1] "a"
[1] "b"
[1] "c"
[1] "d"
```

for loops can be nested inside of each other.

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

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

Nested loops are commonly needed for multidimensional or hierarchical data structures

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, until the condition is false, after which the loop exits.

```
> 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. Use with care!

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

```
> z <- 5
> set.seed(1)
>
> while(z >= 3 && z <= 10) {
+       coin <- rbinom(1, 1, 0.5)
+
+       if(coin == 1) {  ## random walk
+             z <- z + 1
+       } else {
+             z <- z - 1
+       }
+ }
> print(z)
[1] 2
```

repeat initiates an infinite loop right from the start. These are not commonly used in statistical or data analysis applications but they do have their uses. The only way to exit a repeat loop is to call break.One possible paradigm might be in an iterative algorith where you may be searching for a solution and you don’t want to stop until you’re close enough to the solution. In this kind of situation, you often don’t know in advance how many iterations it’s going to take to get “close enough” to the solution.

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

repeat {
	x1 <- computeEstimate()

	if(abs(x1 - x0) < tol) {  ## Close enough?
		break
	} else {
		x0 <- x1
	}
}
```

next is used to skip an iteration of a loop.

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

break is used to exit a loop immediately, regardless of what iteration the loop may be on.

```r
for(i in 1:100) {
    print(i)

    if(i > 20) {
        ## Stop loop after 20 iterations
        break
    }
}
```