Control Structures

Control the flow of an program. Depending on the 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 expressions.

If-else

Structure:

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

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

Example

```
if(x >3){
    y <- 10
} else {
    y <- 0
}
## assigning a valye to y
y <- if(x > 3) {
    10
} else {
        0
}
```

For Loops

`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

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

The loop take the `i` variable and in each function of the loop gives it values 1..10 and then exit(`i` is index variable)

These 3 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)) { ## input vector, create a integer sequence = length of the vector
    print(x[1])
}

for(letter in x) { ## don't have to be an integer
    print(letter)
}

for(i in 1:4) print(x[1]) ## no need brace when single expression
```

For loops can be nested

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

for(i in seq_len(nrow(x))) { ## create integer sequence 1 to 2
    for(j in seq_len(ncol(x))) { ## create integer sequence 1 to 3
        print(x[i, j])
    }
}
```

2 or 3 levels are appropriated in the nesting

While Loops

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. (infinite, so must have a stop, for safety, sometimes `for` is a better option)

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

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) ## an even coin

    if(coin == 1) {## random walk, value of z depends on the coin
        x <- -z + 1
    } else {
        z <- -z -1
    }
}
```

It is hard to tell when will finish(contain the generation of the random number, so it will stop when `z =10` or `z <= 3`)

Conditions are always evaluated from left to right, if the both condition are true, the program will continue running.

Repeat Next Break

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

```
x0 <- 1
tol <- 1e-8 ## tolerance is 10^-8
repeat {
    x1 <- computeEstimate() ## not a real function

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

The precious one is a bit dangerous since there's no stop in it. It is better to set a hard limit on the number of iterations(using a loop) and then report whether convergence was achieved or not.

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

```
for (i in 1:100) {
    if(i >= 20) {
        ## skip the fitst 10 iteration
        next
    }
    ## do something here
}
```

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

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 program, for command-line interactive work, the *apply functions are more helpful