Loops in R

R has three statements that provide explicit looping. They are **`for`**, **`while`** and **`repeat`**. The two built-in constructs, `next` and `break`, provide additional control over the evaluation. Each of the three statements returns the value of the last statement that was evaluated. It is possible, although uncommon, to assign the result of one of these statements to a symbol. R provides other functions for implicit looping such as `tapply`, `apply`, and `lapply`. In addition many operations, especially arithmetic ones, are vectorized so you may not need to use a loop.

There are two statements that can be used to explicitly control looping. They are `break` and `next`. The `break` statement causes an exit from the innermost loop that is currently being executed. The `next` statement immediately causes control to return to the start of the loop. The next iteration of the loop (if there is one) is then executed. No statement below `next` in the current loop is evaluated.

The value returned by a loop statement is always `NULL` and is returned invisibly.

1. `repeat` loop

The **`repeat`** statement causes repeated evaluation of the body until a break is specifically requested. This means that you need to be careful when using **`repeat`** because of the danger of an infinite loop. The syntax of the **`repeat`** loop is

`repeat` *`statement`*

When using **`repeat`**, *statement* must be a block statement. You need to both perform some computation and test whether or not to break from the loop and usually this requires two statements.

Example:

Program #1*: *Print sequence of 'n' numbers

```
n <- 0
repeat
{
n <- n + 1
if(n > 10)
{
break
}
print(n)
}
```

***Output*:**

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
```

```
[1] 8
[1] 9
[1] 10
```

2. `while` loop

The **`while`** statement is very similar to the **`repeat`** statement. The syntax of the **`while`** loop is

```
while ( statement1 ) statement2
```

where *statement1* is evaluated and if its value is `TRUE` then *statement2* is evaluated. This process continues until *statement1* evaluates to `FALSE`.

Example:

Program #2*: *Generating Fibonacci sequence upto 'n' number

```
n <- 10
Fib1 <- 1
Fib2 <- 1
Fib <- Fib1
while (Fib2 < n)
{
Fib <- c(Fib, Fib2)
Temp <- Fib2
Fib2 <- Fib1 + Fib2
Fib1 <- Temp
}
Fib
```

***Output*:**

```
> Fib
[1] 1 1 2 3 5 8
```

3. `for` loop

The syntax of the **`for`** loop is

```
for ( name in vector )
statement1
```

where *vector* can be either a vector or a list. For each element in *vector* the variable *name* is set to the value of that element and *statement1* is evaluated. A side effect is that the variable *name* still exists after the loop has concluded and it has the value of the last element of *vector* that the loop was evaluated for.

Example:

Program #3*: *Generating Fibonacci sequence for 'n' numbers

```
n <- 10
fib <- numeric(n)
fib[1] <- 1
fib[2] <- 1
for (i in 3:n)
{
fib[i] <- fib[i-1]+fib[i-2]
```

```
}
fib
```

***Output*:**

```
> fib
 [1]  1  1  2  3  5  8 13 21 34 55
```