LAB100

Week 9: Looping

Your aim for this workshop is to learn how to simplify your code by using loops to replace repetitive calculations. Upon completing this workshop you should be able to:

- Construct `for` and `while` loops
- Understand the difference between `for` and `while` loops
- Use if/else statements within loops
- Use nested for loops
- Cope with infinite loops.

Remember that you need to change the working directory and open a new R script at the start of each workshop.

1 Constructing `for` Loops

Lets begin with an example: calculate the 5^{th} number in the Fibonacci sequence. It begins with the initial values $x_1 = 1$ and $x_2 = 1$ and the remaining elements of the sequence, for $n = 3, 4, \ldots$, is defined by the recursive equation:

$$x_n = x_{n-1} + x_{n-2}$$

We can calculate this by executing the code:

```
x <- rep(1, 5)
x[3] <- x[2] + x[1]
x[4] <- x[3] + x[2]
x[5] <- x[4] + x[3]
x
```

Now, what is the 20^{th} number of the Fibonacci sequence? Writing numerous versions of the recursive formula can be extremely tedious and can be very difficult to identify typing errors. We can simplify the code by using a `for` loop that repeats a small block of code a specified number of times. The general structure for a `for` loop in RStudio is:

```
for(<LOOP.VAR> in <RANGE>){
  <EXPRESSION>
}
```

The command begins by the definition of a variable <LOOP.VAR> that will sequentially take each value in the vector <RANGE> for each loop. During each loop the code <EXPRESSION> is evaluated and it may depend on the looping variable.

We can construct a for loop to calculate the Fibonacci sequence. Type and run the following example:

```
MAX <- 5
x <- rep( 1, MAX )
for( n in 3:MAX){
   x[n] <- x[n-1] + x[n-2]
}
x
```

This example calculates the first five values of the sequence. It begins by creating the vector x that will store the sequence and defining the first and second elements according to the sequence's initial values. The for loop starts by assigning the looping variable n the first value in the range, i.e. 3. It then executes the command that adds the first and second elements of the vector x and assigns the value to the vector's third element. At the next iteration, the looping variable is assigned the next value in the range 3:MAX and the recursive command is performed again. This continues until n has reached the end of the looping range.

2 Calculation with the looping variable

In the previous example the looping variable was used as a pointer that extracts and inserts data from a vector. We can also use this variable within the calculations of the loop. For example, the factorial for a non-negative number, n, has the following recursive calculations:

$$n! \;=\; n \times (n-1)!$$

Using the initial definition $0! = 1$, a for loop can be constructed to evaluate factorials. For example, type and execute the following code to calculate $6!$.

```
MAX <- 6
x <- 1              # Initial value 0!  = 1
for( n in 1:MAX ){
```

```
   x <- n*x          # Recursive formula
}
x
```

Check your answer is equal to `factorial(6)`.

3 If/else Statements Within Loops

Consider the Fibonacci sequence from earlier in this workshop, but with the additional constraint that whenever a number in the sequence id evaluated to be greater than or equal to 10, then we subtract 10 from this number.

To understand what we need within the `for` loop we need to examine what our adapted Fibonacci sequence is doing for a particular loop, indexed by `n`. The first step is to evaluate the Fibonacci recursion and store the result in the temporary variable `y`:

$$y \quad = \quad x_{n-1} + x_{n-2}$$

If the condition $x \geq 10$ is satisfied, the we determine the next value in the sequence to be:

$$x_n \quad = \quad y - 10$$

Otherwise, the next value is:

$$x_n \quad = \quad y$$

Now that we understand what is happening within one iteration, we can construct a `for` loop where the looping variable `n` sequentially takes each value in the looping range. Type the following into your editor and execute to find the 10^{th} value in our adapted Fibonacci sequence.

```
MAX <- 10
x <- rep(1,MAX)
for( n in 3:MAX ){
   y <- x[n-1] + x[n-2]
   if( y >= 10 ){
      x[n] <- y - 10
   }
   else{
      x[n] <- y
   }
}
x
```

4 Nested for loops

In some situations it will be useful or necessary to loop over all combinations of several different variables. For this task we may need to use nested `for` loops where one (or more) loop appears inside another. As a simple example, we could consider determining the frequencies of different outcomes from rolling a pair of dice. We can let `i1` represent the result of the roll for the first die and `i2` represent the result of the roll for the second die. A crude way of counting the number of ways in which the dice sum to 10 or more is to perform the following code

```
counter <- 0
for (i1 in 1:6) {
   for (i2 in 1:6) {
      if (i1 + i2 >= 10) {
         counter <- counter + 1
      }
   }
}
counter
```

5 While Loops

So far we have only considered creating loops that have a known number of iterations. However, there are some algorithms that continue to loop through the same set of calculations whilst a condition is still satisfied.

Persisting with the Fibonacci sequence example, suppose we wish to find the value of the first Fibonacci number that is greater than 500. We could go about this by making a sufficently long vector, `x`, of Fibonacci numbers using a `for` loop and then using

```
min(x[x > 500])
```

However this only works if we know how long the vector `x` needs to be, and the code may be inefficient if we inadvertently calculate too long a sequence. Instead, we can use a `while` loop and only continue computing new Fibonacci number **while** $x_n \leq 500$ (so that the final number we compute will be the first > 500.

```
x <- rep( 1, 2)
n <- 2
while( x[n] <= 500){
   x[n+1] <- x[n] + x[n-1]
```

```
  n <- n +1
}
x[n]
```

The general structure for a `while` loop is as follows:

```
while( <CONDITION> ){
  <EXPRESSION>
}
```

6 Warning of Infinite Loops

Note that a `while` loop will only stop when the `<CONDITION>` stops being true. If we are not careful we can quickly reach a situation where the loop will continue indefinitely. For instance if, in the Fibonacci sequence example above, we changed the condition to `while(x[n] > 0)` we would never reach a value of `n` for which this was not met. If you find yourself in an infinite loop (or what appears to be an infinite loop) situation, click on the STOP-sign button at the top of the console panel.

To avoid code going into infinite loops it is necessary to attempt to investigate all possible circumstances for all potential initial values *before* writing your code.

Quiz 1: Recursion

Construct a `for` loop that performs the following recursion:

$$x_n = \left(n - \frac{1}{2}\right) \times x_{n-1}$$

Correct to 2 decimal places, what is x_7 when the initial value is $x_0 = \sqrt{\pi}$?

Quiz 2: If/else statement within For Loop

Construct an if/else statement within a `for` loop for the recursive formula:

$$x_n = \begin{cases} x_{n-1} - n & \text{if } x_{n-1} \geq 4 \\ x_{n-1} + 1 & \text{otherwise} \end{cases}$$

Find the largest $n \leq 100$ such that $x_n = 4$.

Quiz 3: Nested For Loops

Suppose I have the following numbers of coins of different denominations:

Coin	2p	5p	10p	20p	50p
Frequency	15	5	7	5	2

What is the number of distinct ways in which I could make exactly £1 from subsets of those coins?
[For instance (5× 2p, 2× 5p and 4× 20p) would be one possible way.]
(**A**) 52 (**B**) 72 (**C**) 82 (**D**) 92 (**E**) 102

Quiz 4: Newton-Raphson method (i)

One procedure for which a `while` loop is useful for implementation is the Newton-Raphson method for finding the roots of the differentiable function $f(x)$. This algorithm iterates the following recursion

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})},$$

whilst the difference between successive evaluations is greater than a specified small value. Consider using the Newton-Raphson method to find one of the roots of the function:

$$f(x) \quad = \quad x^4 - 4x^2 - \frac{36}{7}$$

Take $x_0 = -4$ as the your starting point and use

$$|x_n - x_{n-1}| \leq 1 \times 10^{-5}$$

as the convergence criterion.
Give the value of x_n at convergence to 2 decimal places.

Quiz 5: Newton-Raphson method (ii)

Determine the number of iterations needed to reach convergence i.e. the first n for which

$$|x_n - x_{n-1}| \leq 1 \times 10^{-5}$$
