Control flow in R

Control structures in R allow you to control the flow of execution of a script (typically inside of a function).

- `if` and `if()...else`: used for conditional statements,
- `for()`: used for a defined number of repetitions,
- `while()`: used to run a code until a condition is met.

These are standard to most programming languages.

Note:

- There is usually more than one way to control the flow in R.
- Using loops (*i.e.*, `for()` and `while()`) is often not the quickest way, either to code or to run.

if() and if()...else

Basic syntax for `if()` and `if()...else`:

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

For example, we can check when a value is negative or not.

```
> x <- 5
> if (x < 0) {print("x is negative")
+   } else {print("x is non-negative")}
[1] "x is non-negative"
```

if(), if()...else and ifelse()

Basic syntax for `if()` and `if()...else`:

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

Vectorization with `ifelse`: *Alternative w (shortcut)*

```
> x <- 5
> if (x<0) {print("x is negative")
+   } else {print("x is non-negative")}
[1] "x is non-negative"
> ifelse(x<0, "x is negative", "x is non-negative")
[1] "x is non-negative"
```

if()

Example 1: The condition in brackets must be a single *logical* value (TRUE or FALSE).

```
> x  <-  4 == 3
> if (x) {"4 equals 3"}
# No message will be printed because the vector x is FALSE.
```

Example 2: The condition in brackets must be a *single* logical value (TRUE or FALSE).

```
> x  <-  c(1,5)
> y  <-  x %in% c(3, 5, 16)
> y
[1] FALSE  TRUE
> if (y) {
+   "at least one number in x was found within the sequence c(3, 5, 16)"}
Warning message:
In if (y) { :
  the condition has length > 1 and only the first element will be used
```

if() with any() and all()

```
> x  <-  c(1,5)
> y  <-  x %in% c(3, 5, 16)

> any(y) # given a set of logical vectors, is at least one of the values true?
[1] TRUE
> if(any(y)) {
+ "at least one number in x was found within the sequence c(3, 5, 16)"}
[1] "at least one number in x was found within the sequence c(3, 5, 16)"

> all(y) # given a set of logical vectors, are all of the values true?
[1] FALSE
> if(all(y)) {
+ "all numbers in x were found within the sequence c(3, 5, 16)"}
```

Exercise

Task

Write an `R` function which:

1. Takes as argument a vector `x`.
2. Returns `x` if all the elements of `x` are non-negative. Else, it returns a message saying that "Some elements in the input vector are negative".

For example, suppose your input vector is `x=(1,0,1,5,100)`. Then the function should return `x`. If your input vector is `x=(1,-1,1)`, then the function should return "Some elements in the given vector are negative".

for() loop

Basic syntax for `for()`:

```
for(iterator in set of values){
  do a thing
}
```

Example:

```
> iter <- 3
> for(i in 1:iter){
+   print(letters[i])
+ }
[1] "a"
[1] "b"
[1] "c"
```

`for()` and execution time

One of the most important issues with `for()` loops is execution time. Consider the following lines of code. Changing the value of `n` will affect the time the computer needs to run the code. But how do we find out how long it takes to run the code for `n`=10 vs `n`=1000000?

```
> n <- 10
> vec1  <-  numeric(length=n); vec2  <-  numeric(length=n)
> vec  <-  numeric(length=n)
> for(i in 1:n) {
+   vec1[i]  <-  rnorm(1)
+   vec2[i]  <-  rnorm(1)
+   vec[i] <- vec1[i]+vec2[i]
+ }
> vec
 [1] -1.85658283  0.08776951 -1.99049482 -0.90179469  0.25836012
 [6] -1.36453494 -0.33188221  1.25939388 -0.89130908  0.08318155
```

system.time()

The system.time() function is a useful way of finding out how long something takes to run.

```
> n <- 1000000
> vec1  <-  numeric(length=n); vec2  <-  numeric(length=n)
> vec  <-  numeric(length=n)
> system.time(
+   for(i in 1:n) {
+     vec1[i]  <-  rnorm(1)
+     vec2[i]  <-  rnorm(1)
+     vec[i] <- vec1[i]+vec2[i]
+   }
+ )
   user  system elapsed
  2.50    0.01    2.51
```

system.time()

The proc.time() function can also be used, by defining a starting point at the beginning of your main code of function, and recall it at the end to count how long passed. The value returned by proc.time() is the number of seconds since you opened your R session.

```
> n <- 1000000
>  vec1  <-  numeric(length=n); vec2  <-  numeric(length=n)
>  vec  <-  numeric(length=n)
>  start=proc.time()
>    for(i in 1:n) {
+      vec1[i]  <-  rnorm(1)
+      vec2[i]  <-  rnorm(1)
+      vec[i] <- vec1[i]+vec2[i]
+    }
> proc.time()-start
   user  system elapsed
   2.51    0.03    2.56
```

for() and execution time

Compare it with

```
> n <- 1000000
> vec  <-  numeric(length=n)
> system.time(
+   for(i in 1:n) {
+     vec[i] <- rnorm(1)+rnorm(1)
+   })
   user  system elapsed
  2.223   0.397   2.625
```

or even

```
> system.time(
+   v <- rnorm(n)+rnorm(n))
   user  system elapsed
  0.099   0.000   0.104
```

Nested `for()` loops

A **nested** `for()` loop is a for statement inside another for statement. In a nested `for()` loop, one iteration of the outer loop is first executed, after which the inner loop is executed. Once the inner loop is executed, the program moves to the next iteration of the outer loop, *etc.*

```
> m <- matrix(1:6, 2)
> for (i in seq(nrow(m))) {
+   for (j in seq(ncol(m))) {
+     print(m[i, j])
+   }
+ }
[1] 1
[1] 3
[1] 5
[1] 2
[1] 4
[1] 6
```

`while()` loop

Sometimes you need to repeat an operation until a certain condition is met (rather than doing it for a specific number of times). In such cases the `while()` loop is useful.

Basic syntax for `while()`:

```
while(this condition is true){
  do a thing
}
```

Example:

Note that you have to give a value to the variable i in the while condition before you use it in the logical expression (i<3). Note that this is different from for loop, which defines the range of the counter, e.g. for (i in 1:3)

```
> i <- 1
> while (i < 3) {
+   print(i)
+   i <- i + 1
+ }
[1] 1
[1] 2
```

`while()` loop

We want to generate random numbers between 0 and 1 until we get one that is at most 0.5. So we try the code below.

```
> while(z > 0.5){
+   z <- runif(1)
+   print(z)
+ }
Error in z : object 'z' not found
```

But this doesn't work! Can you see why?

```
> z <- 1
> while(z > 0.5){
+   z <- runif(1)
+   print(z)
+ }
[1] 0.4177174
```

Task

In mathematics, the Fibonacci numbers (commonly denoted by F_n) form a sequence, called the **Fibonacci sequence**, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is,

$$\begin{aligned} &F_1 = 0, \quad F_2 = 1 \\ &F_n = F_{n-1} + F_{n-2} \quad \text{for} \quad n \geq 3. \end{aligned}$$

1. Using a `for()` or a `while()` loop, produce a vector containing the first 20 values in the Fibonacci sequence.
2. Write a function that takes as input the argument `n` and gives as output a vector of length `n`, which contains the first `n` numbers of the Fibonacci sequence.