Topic 3: Control Structures in R

ISOM3390: Business Programming in R

Overview of Control Structures

- Control structures in R allow you to control the flow of execution of the program, depending on runtime conditions. Common structures include

    - `if-else`: test 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 iteration of a loop

- Most control structures are not used in interactive sessions, but rather when writing functions or longer expressions.

Conditionals: `if` and `else`

```
if (<condition_1>) {
            # do something
} else if (<condition_2>) {
            # do something different
} else {
            # do something different
}
```

- Conditions need to give **one** `TRUE` or `FALSE` value
- The `else if` clause and `else` clause are not necessary.
- Can be written in a variety of forms, e.g., one-line actions don't need braces

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

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

- Can nest arbitrarily deeply:

```
if (x ^ 2 < 1) {
    x ^ 2
} else {
    if (x >= 0) {
        2 * x - 1
    } else {
        - 2 * x - 1
    }
}
```

Combining Logicals

- Conditional execution requires a **single** logical value.
- unlike `&` and `|` combine logical values element-wise, `&&` and `||` give one logical value, **lazily**:

```
(0 > 0) && (z <- (9 > 4))
```

```
## [1] FALSE
```

```
exists("z")
```

```
## [1] FALSE
```

```
(1 > 0) && (z <- (9 > 4))
```

```
## [1] TRUE
```

```
exists("z")
```

```
## [1] TRUE
```

MuLtiple Selection: `switch()`

```
cars$speed
```

```
##  [1]  4  4  7  7  8  9 10 10 10 11 11 12 12 12 12 13 13 13 13 14 14 14 14
## [24] 15 15 15 16 16 17 17 17 18 18 18 18 19 19 19 20 20 20 20 20 22 23 24
## [47] 24 24 24 25
```

```
if(type.of.summary=="mean") {
  mean(cars$speed)
  } else if(type.of.summary=="median") {
    median(cars$speed)
    } else if (type.of.summary=="histogram") {
      hist(cars$speed)
      } else "I don't understand"
```

Simplify nested `if-else` with `switch()`

```
# A variable is given to select on; then a value is assigned to each option
switch(type.of.summary, mean = mean(cars$speed), median = median(cars$speed),
       histogram=hist(cars$speed), "I don't understand")
```

Loops: `for`

A `for` loop takes an iterator variable (a counter) and assign it successive values from a sequence or a vector; most commonly used for iterating over the elements of an object (a list, a vector, etc.)

```r
x <- c("a","b","c","d")
for(i in 1:4) print(x[i])
```

```
## [1] "a"
## [1] "b"
## [1] "c"
## [1] "d"
```

```r
for(i in seq_along(x)) {
    print(x[i])
}
```

```
## [1] "a"
## [1] "b"
## [1] "c"
## [1] "d"
```

```
for(letter in x) print(letter)
```

```
## [1] "a"
## [1] "b"
## [1] "c"
## [1] "d"
```

- `for` is used when the number of times to repeat (values to iterate over) is clear in advance.

Nested `for` loops

`for` loops can be nested:

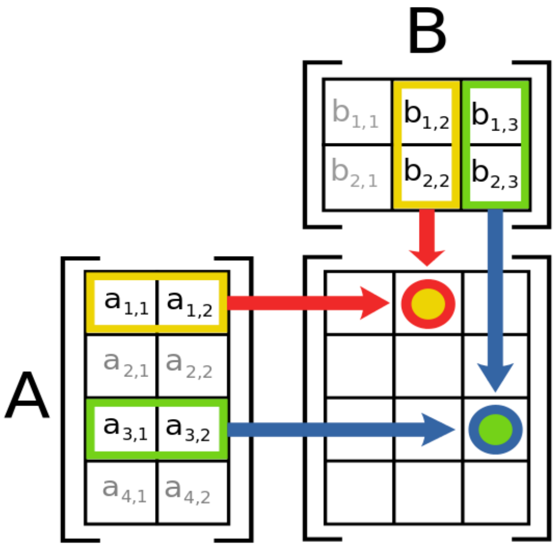
```r
(x <- matrix(1:6, 2, 3))
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

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

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

An Nested Loop Example: Matrix Multiplication



A triple `for` loop to implement it:

```
C <- matrix(0, nrow=nrow(A), ncol=ncol(B))
if (ncol(A) == nrow(B)) {
    for (i in 1:nrow(C)) {
         for (j in 1:ncol(C)) {
              for (k in 1:ncol(A)) {
                   C[i,j] <- C[i,j] + A[i,k]*B[k,j]
              }
         }
     }
} else {
    stop("matrices a and b non-conformable")
}
```

Loops: `while`

- A `while` loop begins by testing a condition. If it's `TRUE`, then it executes the loop body. Once the loop body is executed, the condition is tested again, and so on.

- Conditions used by `while` must be a **single** logical value (like `if`)

```
count <- 0
while(count < 5) {
 print(count)
 count <- count + 1
}
```

```
## [1] 0
## [1] 1
## [1] 2
## [1] 3
## [1] 4
```

- `while` is used when we can recognize when to stop once we're there, even if we can't guess it to begin with.

Unconditional Loops: `repeat`

- `repeat` initiates an infinite loop. It will execute until we press `Escape` or quit R, whichever happens soonest.

- The infinite loop can be broken out of by introducing a `break` statement via a conditional.

```
x0 <- 1
tol <- 1e-8
repeat {
  x1 <- computeEstimate()
  if(abs(x1 - x0) < tol) break
  else x0 <- x1
  }
```

- A `next` statement skips the rest of the current iteration and starts the next iteration.

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

Vectorized Operations

Consider implementing a vector addition $a + b$ using iteration:

```r
c <- vector("numeric", length(a))
for (i in seq_along(a)) c[i] <- a[i] + b[i]
```

How R adds 2 vectors and does matrix multiplication:

```r
a + b
A %*% B
```

Vectorization eliminates many loops.

Advantages of Vectorization

In R, **vectorization** means operations or functions operate on all elements of a vector without needing to loop through and act on each element one at a time.

- Clarity: the syntax is about **what** we're doing
- Abstraction: the syntax hides **how** the computer does it
- Concision: we write less
- Generality: same syntax works for numbers, vectors, arrays, …
- Speed: modifying big vectors over and over is slow in R; work gets done by optimized low-level code

Vectorized Conditionals: `ifelse()`

The 1st argument takes a logical vector.

Depending on `TRUE` or `FALSE`, it picks elements for the returned value from either the 2nd or 3rd arguments:

```
(x<-runif(6, 0, 2))
```

```
## [1] 1.77715344 0.09222125 1.45423310 1.27006517 0.13902005 0.66145200
```

```
ifelse (x ^ 2 > 1, 2 * abs(x) - 1, x ^ 2)
```

```
## [1] 2.55430689 0.00850476 1.90846620 1.54013033 0.01932657 0.43751875
```

```
# it can also accept vectors in the second and third arguments.
# The recycling rule applies when the vectors aren't the same size.
ifelse (x ^ 2 > 1, 1:3, -1:-6)
```

```
## [1]  1 -2  3  1 -5 -6
```

Summary

- Control structures like `if`, `while`, and `for` allow us 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 programs; for command-line interactive work, the `apply` functions are more useful (later).
- Avoiding iteration with whole-object ("vectorized") operations.