Lecture 17: Control Stuctures

STAT 450, Fall 2021

Control structures allow you to control the flow of execution of R code. There are two primary types of control structures: `if-else` statements and `for` loops.

`if` statement

The syntax of an `if` statement:

```r
if(condition) {
  statement1
}
```

First, the `condition` is evaluated. If it is `TRUE`, then `statement1` is evaluated; otherwise, nothing is done. The `condition` is a logical expression.

Example:

```r
x <- 7
if(x >= 0 & x <= 10) {
  print("Number is between 0 and 10")
}
```

```
## [1] "Number is between 0 and 10"
```

```r
x <- 47
if(x >= 0 & x <= 10) {
  print("Number is between 0 and 10")
}
```

if-else statement

The syntax of an `if-else` statement:

```
if(condition) {
  statement1
} else {
  statement2
}
```

First the `condition` is evaluated. If it is `TRUE`, then `statement1` is evaluated; otherwise `statement2` is evaluated.

We can generalize this syntax to evaluate more than one condition:

```
if(condition) {
  statement1
} else if(condition) {
  statement2
} else {
  statement3
}
```

Example: Flipping a coin.

```
u <- runif(1) # generate random number between 0 and 1
u
```

```
## [1] 0.8241875
```

```
if(u > 0.5) {
  print("Heads")
} else {
  print("Tails")
}
```

```
## [1] "Heads"
```

Example: Assigning a grade.

```
x <- 91
if(x > 90) {
  print("A")
} else if(x > 80) {
  print("B")
} else if (x > 70) {
  print("C")
} else {
  "NC"
}
```

```
## [1] "A"
```

`for` loops

A `for` loop iterates over elements of a vector. A `for` loop is useful when performing the same operation repeatedly. Here are some simple examples:

```
for(i in 1:5) {
  print(i)
}
```

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

```
for(x in 1:5) {
  print(paste(x, "squared is", x^2))
}
```

```
## [1] "1 squared is 1"
## [1] "2 squared is 4"
## [1] "3 squared is 9"
## [1] "4 squared is 16"
## [1] "5 squared is 25"
```

Suppose we want to compute the mean of each column of the `mtcars` data frame.

```
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

We could copy-and-paste the code, modifying the column index each time. But this becomes tedious:

```
mean(mtcars[, 1])
```

```
## [1] 20.09062
```

```
mean(mtcars[, 2])
```

```
## [1] 6.1875
```

```
mean(mtcars[, 3])
```

```
## [1] 230.7219
```

```
# and so on
```

A better way is to use a loop:

```r
avgs <- c()
for(i in 1:ncol(mtcars)) {
  avgs[i] <- mean(mtcars[, i])
}
avgs
```

```
##  [1]  20.090625   6.187500 230.721875 146.687500   3.596563   3.217250
##  [7]  17.848750   0.437500   0.406250   3.687500   2.812500
```

We can include the variable names to improve presentation:

```r
names(avgs) <- names(mtcars)
avgs
```

```
##        mpg        cyl       disp         hp       drat         wt       qsec
##  20.090625   6.187500 230.721875 146.687500   3.596563   3.217250  17.848750
##         vs         am       gear       carb
##   0.437500   0.406250   3.687500   2.812500
```

apply()

An alternative way to do this is with the `apply()` function:

```r
apply(mtcars, MARGIN = 2, FUN = mean)
```

```
##        mpg        cyl       disp         hp       drat         wt       qsec
##  20.090625   6.187500 230.721875 146.687500   3.596563   3.217250  17.848750
##         vs         am       gear       carb
##   0.437500   0.406250   3.687500   2.812500
```

Here, the `apply()` function was used to evaluate the `mean` function over each column of `mtcars`. Note that setting `MARGIN = 2` evaluated the function over the columns. Setting `MARGIN = 1` would evaluate the function over the rows.

Exercise: Use a `for` loop to simulate flipping a coin 10 times.

Exercise: Compute the standard deviation of each column of `mtcars` by using:

(1) a `for` loop

(2) the `apply` function