8.2: Loops

The description I gave earlier for how a script works was a tiny bit of a lie. Specifically, it's not necessarily the case that R starts at the top of the file and runs straight through to the end of the file. For all the scripts that we've seen so far that's exactly what happens, and unless you insert some commands to explicitly alter how the script runs, that is what will *always* happen. However, you actually have quite a lot of flexibility in this respect. Depending on how you write the script, you can have R repeat several commands, or skip over different commands, and so on. This topic is referred to as ***flow control***, and the first concept to discuss in this respect is the idea of a ***loop***. The basic idea is very simple: a loop is a block of code (i.e., a sequence of commands) that R will execute over and over again until some termination criterion is met. Looping is a very powerful idea. There are three different ways to construct a loop in R, based on the `while`, `for` and `repeat` functions. I'll only discuss the first two in this book.

8.2.1 `while` loop

A `while` loop is a simple thing. The basic format of the loop looks like this:

```
while ( CONDITION ) {
       STATEMENT1
       STATEMENT2
       ETC
    }
```

The code corresponding to CONDITION needs to produce a logical value, either `TRUE` or `FALSE`. Whenever R encounters a `while` statement, it checks to see if the CONDITION is `TRUE`. If it is, then R goes on to execute all of the commands inside the curly brackets, proceeding from top to bottom as usual. However, when it gets to the bottom of those statements, it moves back up to the `while` statement. Then, like the mindless automaton it is, it checks to see if the CONDITION is `TRUE`. If it is, then R goes on to execute all … well, you get the idea. This continues endlessly until at some point the CONDITION turns out to be `FALSE`. Once that happens, R jumps to the bottom of the loop (i.e., to the `}` character), and then continues on with whatever commands appear next in the script.

To start with, let's keep things simple, and use a `while` loop to calculate the smallest multiple of 17 that is greater than or equal to 1000. This is a very silly example since you can actually calculate it using simple arithmetic operations, but the point here isn't to do something novel. The point is to show how to write a `while` loop. Here's the script:

```
## --- whileexample.R
x <- 0
while ( x < 1000 ) {
  x <- x + 17
}
print( x )
```

When we run this script, R starts at the top and creates a new variable called `x` and assigns it a value of 0. It then moves down to the loop, and "notices" that the condition here is `x < 1000`. Since the current value of `x` is zero, the condition is true, so it enters the body of the loop (inside the curly braces). There's only one command here[135] which instructs R to increase the value of `x` by 17. R then returns to the top of the loop, and rechecks the condition. The value of `x` is now 17, but that's still less than 1000, so the loop continues. This cycle will continue for a total of 59 iterations, until finally `x` reaches a value of 1003 (i.e., 59×17=1003). At this point, the loop stops, and R finally reaches line 5 of the script, prints out the value of `x` on screen, and then halts. Let's watch:

```
source( "./rbook-master/scripts/whileexample.R" )
```

```
## [1] 1003
```

Truly fascinating stuff.

8.2.2 `for` loop

The `for` loop is also pretty simple, though not quite as simple as the `while` loop. The basic format of this loop goes like this:

```
for ( VAR in VECTOR ) {
       STATEMENT1
       STATEMENT2
       ETC
    }
```

In a `for` loop, R runs a fixed number of iterations. We have a VECTOR which has several elements, each one corresponding to a possible value of the variable VAR. In the first iteration of the loop, VAR is given a value corresponding to the first element of VECTOR; in the second iteration of the loop VAR gets a value corresponding to the second value in VECTOR; and so on. Once we've exhausted all of the values in VECTOR, the loop terminates and the flow of the program continues down the script.

Once again, let's use some very simple examples. Firstly, here is a program that just prints out the word "hello" three times and then stops:

```
## --- forexample.R
for ( i in 1:3 ) {
  print( "hello" )
}
```

This is the simplest example of a `for` loop. The vector of possible values for the `i` variable just corresponds to the numbers from 1 to 3. Not only that, the body of the loop doesn't actually depend on `i` at all. Not surprisingly, here's what happens when we run it:

```
source( "./rbook-master/scripts/forexample.R" )
```

```
## [1] "hello"
## [1] "hello"
## [1] "hello"
```

However, there's nothing that stops you from using something non-numeric as the vector of possible values, as the following example illustrates. This time around, we'll use a character vector to control our loop, which in this case will be a vector of `words`. And what we'll do in the loop is get R to convert the word to upper case letters, calculate the length of the word, and print it out. Here's the script:

```
## --- forexample2.R

#the words_
words <- c("it","was","the","dirty","end","of","winter")

#loop over the words_
for ( w in words ) {

  w.length <- nchar( w )     # calculate the number of letters_
  W <- toupper( w )          # convert the word to upper case letters_
  msg <- paste( W, "has", w.length, "letters" )   # a message to print_
  print( msg )               # print it_

}
```

And here's the output:

```
source( "./rbook-master/scripts/forexample2.R" )
```

```
## [1] "IT has 2 letters"
## [1] "WAS has 3 letters"
## [1] "THE has 3 letters"
## [1] "DIRTY has 5 letters"
## [1] "END has 3 letters"
## [1] "OF has 2 letters"
## [1] "WINTER has 6 letters"
```

Again, pretty straightforward I hope.

8.2.3 more realistic example of a loop

To give you a sense of how you can use a loop in a more complex situation, let's write a simple script to simulate the progression of a mortgage. Suppose we have a nice young couple who borrow $300000 from the bank, at an annual interest rate of 5%. The mortgage is a 30 year loan, so they need to pay it off within 360 months total. Our happy couple decide to set their monthly mortgage payment at $1600 per month. Will they pay off the loan in time or not? Only time will tell.[136] Or, alternatively, we could simulate the whole process and get R to tell us. The script to run this is a fair bit more complicated.

```
## --- mortgage.R

# set up
month <- 0        # count the number of months
balance <- 300000 # initial mortgage balance
payments <- 1600  # monthly payments
interest <- 0.05  # 5% interest rate per year
total.paid <- 0   # track what you've paid the bank

# convert annual interest to a monthly multiplier
monthly.multiplier <- (1+interest) ^ (1/12)

# keep looping until the loan is paid off...
while ( balance > 0 ) {

  # do the calculations for this month
  month <- month + 1  # one more month
  balance <- balance * monthly.multiplier  # add the interest
  balance <- balance - payments  # make the payments
  total.paid <- total.paid + payments # track the total paid

  # print the results on screen
  cat( "month", month, ": balance", round(balance), "\n")

} # end of loop

# print the total payments at the end
cat("total payments made", total.paid, "\n" )
```

To explain what's going on, let's go through it carefully. In the first block of code (under `#set up`) all we're doing is specifying all the variables that define the problem. The loan starts with a `balance` of $300,000 owed to the bank on `month` zero, and at that point in time the `total.paid` money is nothing. The couple is making monthly `payments` of $1600, at an annual `interest` rate of 5%. Next, we convert the annual percentage interest into a monthly multiplier. That is, the number that you have to multiply the current balance by each month in order to produce an annual interest rate of 5%. An annual interest rate of 5% implies that, if no payments were made over 12 months the balance would end up being 1.05 times what it was originally, so the *annual* multiplier is 1.05. To calculate the monthly multiplier, we need to calculate the 12th root of 1.05 (i.e., raise 1.05 to the power of 1/12). We store this value in as the `monthly.multiplier` variable, which as it happens corresponds to a value of about 1.004. All of which is a rather long winded way of saying that the *annual* interest rate of 5% corresponds to a *monthly* interest rate of about 0.4%.

Anyway… all of that is really just setting the stage. It's not the interesting part of the script. The interesting part (such as it is) is the loop. The `while` statement on tells R that it needs to keep looping until the `balance` reaches zero (or less, since it might be that the final payment of $1600 pushes the balance below zero). Then, inside the body of the loop, we have two different blocks of code. In the first bit, we do all the number crunching. Firstly we increase the value `month` by 1. Next, the bank charges the interest, so the `balance` goes up. Then, the couple makes their monthly payment and the `balance` goes down. Finally, we keep track of the total amount of money that the couple has paid so far, by adding the `payments` to the running tally. After having done all this number crunching, we tell R to issue the couple with a very terse monthly statement, which just indicates how many months they've been paying the loan and how much money they still owe the bank. Which is rather rude of us really. I've grown attached to this couple and I really feel they deserve better than that. But, that's banks for you.

In any case, the key thing here is the tension between the increase in `balance` on and the decrease. As long as the decrease is bigger, then the balance will eventually drop to zero and the loop will eventually terminate. If not, the loop will continue forever! This is actually very bad programming on my part: I really should have included something to force R to stop if this goes on too long. However, I haven't shown you how to evaluate "if" statements yet, so we'll just have to hope that the author of the book has rigged the example so that the code actually runs. Hm. I wonder what the odds of that are? Anyway, assuming that the loop does eventually terminate, there's one last line of code that prints out the total amount of money that the couple handed over to the bank over the lifetime of the loan.

Now that I've explained everything in the script in tedious detail, let's run it and see what happens:

```
source( "./rbook-master/scripts/mortgage.R" )
```

```
## month 1 : balance 299622
## month 2 : balance 299243
## month 3 : balance 298862
## month 4 : balance 298480
## month 5 : balance 298096
## month 6 : balance 297710
## month 7 : balance 297323
## month 8 : balance 296934
## month 9 : balance 296544
## month 10 : balance 296152
## month 11 : balance 295759
## month 12 : balance 295364
## month 13 : balance 294967
## month 14 : balance 294569
## month 15 : balance 294169
## month 16 : balance 293768
## month 17 : balance 293364
## month 18 : balance 292960
## month 19 : balance 292553
## month 20 : balance 292145
## month 21 : balance 291735
```

```
## month 22 : balance 291324
## month 23 : balance 290911
## month 24 : balance 290496
## month 25 : balance 290079
## month 26 : balance 289661
## month 27 : balance 289241
## month 28 : balance 288820
## month 29 : balance 288396
## month 30 : balance 287971
## month 31 : balance 287545
## month 32 : balance 287116
## month 33 : balance 286686
## month 34 : balance 286254
## month 35 : balance 285820
## month 36 : balance 285385
## month 37 : balance 284947
## month 38 : balance 284508
## month 39 : balance 284067
## month 40 : balance 283625
## month 41 : balance 283180
## month 42 : balance 282734
## month 43 : balance 282286
## month 44 : balance 281836
## month 45 : balance 281384
## month 46 : balance 280930
## month 47 : balance 280475
## month 48 : balance 280018
## month 49 : balance 279559
## month 50 : balance 279098
## month 51 : balance 278635
## month 52 : balance 278170
## month 53 : balance 277703
## month 54 : balance 277234
## month 55 : balance 276764
## month 56 : balance 276292
## month 57 : balance 275817
## month 58 : balance 275341
## month 59 : balance 274863
## month 60 : balance 274382
## month 61 : balance 273900
## month 62 : balance 273416
## month 63 : balance 272930
## month 64 : balance 272442
## month 65 : balance 271952
## month 66 : balance 271460
## month 67 : balance 270966
## month 68 : balance 270470
## month 69 : balance 269972
## month 70 : balance 269472
## month 71 : balance 268970
## month 72 : balance 268465
## month 73 : balance 267959
## month 74 : balance 267451
```

```
## month 75 : balance 266941
## month 76 : balance 266428
## month 77 : balance 265914
## month 78 : balance 265397
## month 79 : balance 264878
## month 80 : balance 264357
## month 81 : balance 263834
## month 82 : balance 263309
## month 83 : balance 262782
## month 84 : balance 262253
## month 85 : balance 261721
## month 86 : balance 261187
## month 87 : balance 260651
## month 88 : balance 260113
## month 89 : balance 259573
## month 90 : balance 259031
## month 91 : balance 258486
## month 92 : balance 257939
## month 93 : balance 257390
## month 94 : balance 256839
## month 95 : balance 256285
## month 96 : balance 255729
## month 97 : balance 255171
## month 98 : balance 254611
## month 99 : balance 254048
## month 100 : balance 253483
## month 101 : balance 252916
## month 102 : balance 252346
## month 103 : balance 251774
## month 104 : balance 251200
## month 105 : balance 250623
## month 106 : balance 250044
## month 107 : balance 249463
## month 108 : balance 248879
## month 109 : balance 248293
## month 110 : balance 247705
## month 111 : balance 247114
## month 112 : balance 246521
## month 113 : balance 245925
## month 114 : balance 245327
## month 115 : balance 244727
## month 116 : balance 244124
## month 117 : balance 243518
## month 118 : balance 242911
## month 119 : balance 242300
## month 120 : balance 241687
## month 121 : balance 241072
## month 122 : balance 240454
## month 123 : balance 239834
## month 124 : balance 239211
## month 125 : balance 238585
## month 126 : balance 237958
## month 127 : balance 237327
```

```
## month 128 : balance 236694
## month 129 : balance 236058
## month 130 : balance 235420
## month 131 : balance 234779
## month 132 : balance 234136
## month 133 : balance 233489
## month 134 : balance 232841
## month 135 : balance 232189
## month 136 : balance 231535
## month 137 : balance 230879
## month 138 : balance 230219
## month 139 : balance 229557
## month 140 : balance 228892
## month 141 : balance 228225
## month 142 : balance 227555
## month 143 : balance 226882
## month 144 : balance 226206
## month 145 : balance 225528
## month 146 : balance 224847
## month 147 : balance 224163
## month 148 : balance 223476
## month 149 : balance 222786
## month 150 : balance 222094
## month 151 : balance 221399
## month 152 : balance 220701
## month 153 : balance 220000
## month 154 : balance 219296
## month 155 : balance 218590
## month 156 : balance 217880
## month 157 : balance 217168
## month 158 : balance 216453
## month 159 : balance 215735
## month 160 : balance 215014
## month 161 : balance 214290
## month 162 : balance 213563
## month 163 : balance 212833
## month 164 : balance 212100
## month 165 : balance 211364
## month 166 : balance 210625
## month 167 : balance 209883
## month 168 : balance 209138
## month 169 : balance 208390
## month 170 : balance 207639
## month 171 : balance 206885
## month 172 : balance 206128
## month 173 : balance 205368
## month 174 : balance 204605
## month 175 : balance 203838
## month 176 : balance 203069
## month 177 : balance 202296
## month 178 : balance 201520
## month 179 : balance 200741
## month 180 : balance 199959
```

```
## month 181 : balance 199174
## month 182 : balance 198385
## month 183 : balance 197593
## month 184 : balance 196798
## month 185 : balance 196000
## month 186 : balance 195199
## month 187 : balance 194394
## month 188 : balance 193586
## month 189 : balance 192775
## month 190 : balance 191960
## month 191 : balance 191142
## month 192 : balance 190321
## month 193 : balance 189496
## month 194 : balance 188668
## month 195 : balance 187837
## month 196 : balance 187002
## month 197 : balance 186164
## month 198 : balance 185323
## month 199 : balance 184478
## month 200 : balance 183629
## month 201 : balance 182777
## month 202 : balance 181922
## month 203 : balance 181063
## month 204 : balance 180201
## month 205 : balance 179335
## month 206 : balance 178466
## month 207 : balance 177593
## month 208 : balance 176716
## month 209 : balance 175836
## month 210 : balance 174953
## month 211 : balance 174065
## month 212 : balance 173175
## month 213 : balance 172280
## month 214 : balance 171382
## month 215 : balance 170480
## month 216 : balance 169575
## month 217 : balance 168666
## month 218 : balance 167753
## month 219 : balance 166836
## month 220 : balance 165916
## month 221 : balance 164992
## month 222 : balance 164064
## month 223 : balance 163133
## month 224 : balance 162197
## month 225 : balance 161258
## month 226 : balance 160315
## month 227 : balance 159368
## month 228 : balance 158417
## month 229 : balance 157463
## month 230 : balance 156504
## month 231 : balance 155542
## month 232 : balance 154576
```

```
## month 233 : balance 153605
## month 234 : balance 152631
## month 235 : balance 151653
## month 236 : balance 150671
## month 237 : balance 149685
## month 238 : balance 148695
## month 239 : balance 147700
## month 240 : balance 146702
## month 241 : balance 145700
## month 242 : balance 144693
## month 243 : balance 143683
## month 244 : balance 142668
## month 245 : balance 141650
## month 246 : balance 140627
## month 247 : balance 139600
## month 248 : balance 138568
## month 249 : balance 137533
## month 250 : balance 136493
## month 251 : balance 135449
## month 252 : balance 134401
## month 253 : balance 133349
## month 254 : balance 132292
## month 255 : balance 131231
## month 256 : balance 130166
## month 257 : balance 129096
## month 258 : balance 128022
## month 259 : balance 126943
## month 260 : balance 125861
## month 261 : balance 124773
## month 262 : balance 123682
## month 263 : balance 122586
## month 264 : balance 121485
## month 265 : balance 120380
## month 266 : balance 119270
## month 267 : balance 118156
## month 268 : balance 117038
## month 269 : balance 115915
## month 270 : balance 114787
## month 271 : balance 113654
## month 272 : balance 112518
## month 273 : balance 111376
## month 274 : balance 110230
## month 275 : balance 109079
## month 276 : balance 107923
## month 277 : balance 106763
## month 278 : balance 105598
## month 279 : balance 104428
## month 280 : balance 103254
## month 281 : balance 102074
## month 282 : balance 100890
## month 283 : balance 99701
## month 284 : balance 98507
## month 285 : balance 97309
```

```
## month 286 : balance 96105
## month 287 : balance 94897
## month 288 : balance 93683
## month 289 : balance 92465
## month 290 : balance 91242
## month 291 : balance 90013
## month 292 : balance 88780
## month 293 : balance 87542
## month 294 : balance 86298
## month 295 : balance 85050
## month 296 : balance 83797
## month 297 : balance 82538
## month 298 : balance 81274
## month 299 : balance 80005
## month 300 : balance 78731
## month 301 : balance 77452
## month 302 : balance 76168
## month 303 : balance 74878
## month 304 : balance 73583
## month 305 : balance 72283
## month 306 : balance 70977
## month 307 : balance 69666
## month 308 : balance 68350
## month 309 : balance 67029
## month 310 : balance 65702
## month 311 : balance 64369
## month 312 : balance 63032
## month 313 : balance 61688
## month 314 : balance 60340
## month 315 : balance 58986
## month 316 : balance 57626
## month 317 : balance 56261
## month 318 : balance 54890
## month 319 : balance 53514
## month 320 : balance 52132
## month 321 : balance 50744
## month 322 : balance 49351
## month 323 : balance 47952
## month 324 : balance 46547
## month 325 : balance 45137
## month 326 : balance 43721
## month 327 : balance 42299
## month 328 : balance 40871
## month 329 : balance 39438
## month 330 : balance 37998
## month 331 : balance 36553
## month 332 : balance 35102
## month 333 : balance 33645
## month 334 : balance 32182
## month 335 : balance 30713
## month 336 : balance 29238
## month 337 : balance 27758
## month 338 : balance 26271
```

```
## month 339 : balance 24778
## month 340 : balance 23279
## month 341 : balance 21773
## month 342 : balance 20262
## month 343 : balance 18745
## month 344 : balance 17221
## month 345 : balance 15691
## month 346 : balance 14155
## month 347 : balance 12613
## month 348 : balance 11064
## month 349 : balance 9509
## month 350 : balance 7948
## month 351 : balance 6380
## month 352 : balance 4806
## month 353 : balance 3226
## month 354 : balance 1639
## month 355 : balance 46
## month 356 : balance -1554
## total payments made 569600
```

So our nice young couple have paid off their $300,000 loan in just 4 months shy of the 30 year term of their loan, at a bargain basement price of $568,046 (since 569600 - 1554 = 568046). A happy ending!
