**Example:**

sim <- 100

# Define a counter variable to determine how many t-values to use

u <- rt(sim, 32)

# Draw sim random t-values, from a t-distribution with 32 degrees of freedom

# Initialize `usq` (could have used a single NA or 0 initialization)

usq <- rep(NA, sim)

print("This loop calculates the square of elements drawn from a t-distribution")

for(i in 1:sim) {

# i-th element of `u1` squared into i-th position of `usq`

usq[i] <- u[i]^2

print(c(u[i], usq[i]))

}

i ## should be 100 (the same as sim), unless the loop had issues!

**Excercise 7:**

Performing the same operation outside of the loop is what is called ***vectorization*** in R. Vectorization is the operation of converting repeated operations on simple numbers (scalars) into single operations on vectors or matrices.

Above we went from squaring every entry in the vector u (operations on scalars) to squaring the vector u (a single operation on a vector). Even though the process of squaring a vector has the same number of operations as the loop, the main benefits are:

* more compact code which is more readable than lengthy loops
* vectors are the base representation in R (a scalar’s inner representation in R is a vector, even though it has only one number).

**Nesting For Loops**

Suppose I wanted to take a matrix and scale it by 10\*sin(2\*pi). To do this I could either

* loop over every, scaling each entry by 10\*sin(2\*pi)
* apply the scaling to the matrix

In this type of operation differences between loops and vectorization become more noticeable.

* When i = j = 7, then the vectorized version is about 7 times faster, but it is barely noticeable to the user.
* When i = j = 100, the difference become more noticeable to the user.
* When i = j = 1000, the loop hangs on for several tens of seconds, when the vectorized version still executes in the blink of an eye.
* When i = j = 1000, the loop takes over a minute and the vectorized version takes about 2.5 seconds.