3. Loops (through a sequence, 0 or more times, or 1 or more times)

Iterate through a sequence (vector or list)

This `for` loop runs `EXPRESSION` for each value `VARIABLE` in `SEQUENCE`. (`UPPER.CASE` text is a placeholder for R code.)

```
for (VARIABLE in SEQUENCE) {
  EXPRESSION
}
```

e.g. Here are canonical sum and product loops. (But use `sum()` and `prod()` when you can.)

```
data = c(2, 3, 5)
total = 0
for (x in data) { # loop through values
  total = total + x
  cat(sep="", "x=", x, ", total=", total, "\n")
}

product = 1
for (i in seq_len(length(data))) { # loop through indices
  product = product * data[i]
  cat(sep="", "data[", i, "]=", data[i], ", product=", product, "\n")
}
```

e.g. Here is one way to find $n!$ ("n factorial"). (But use `factorial()` when you can.)

```
baby.factorial = function(n) {
  stopifnot(n >= 0)
  product = 1
  for (i in seq_len(n)) {
    product = product * i
  }
  return(product)
}
baby.factorial(3)
```

e.g. Here is a non-numeric example.

```
for (file in list.files()) {
  cat(sep="", "file=", file, "\n")
  # ... read file and extract data from it ...
}
```

Loop zero or more times

This `while` loop runs `EXPRESSION` as long as `CONDITION` is true.

```
while (CONDITION) {
  EXPRESSION
}
```

e.g. Show progress of an investment receiving annual compound interest as it grows to $100.

```
balance = 50
interest.rate = 0.07
n.years = 0
while (balance < 100) {
  balance = balance * (1 + interest.rate)
  n.years = n.years + 1
  cat(sep="", "After ", n.years, " years, balance is ", balance, "\n")
}
```

e.g. Here is a second way to find $n! = n(n-1)(n-2)\cdots(3)(2)(1)$.

```
baby.factorial.while = function(n) {
  stopifnot(n >= 0)
  product = 1
  while (n >= 1) {
    product = product * n
    n = n - 1
  }
  return(product)
}
baby.factorial.while(3)
```

Loop one or more times

This loop runs `EXPRESSION` once and then repeats until `CONDITION` is true.

```
repeat {
  EXPRESSION
  if (CONDITION) {
    break
  }
}
```

e.g. Prompt for user input until user cooperates:

```
repeat {
  cat("Please answer 'yes' or 'no':")
  decision = scan(what=character(), n=1, quiet=TRUE) # ?scan
  if ((decision == "yes") | (decision == "no")) {
    break
  }
}
```

Background:

- Loop forever (usually a bad idea):

  ```
  repeat {
     EXPRESSION
  }
  ```

- Break out of a loop with `break`, usually guarded by a condition:

  ```
  if (CONDITION) {
    break
  }
  ```

- Skip to the bottom of a loop (still inside it) with `next`:

  ```
  if (CONDITION) {
    next
  }
  ```

Code formatting tips

- "`{`" does not get a new line
- "`}`" is on a line by itself, indented like the line containing the corresponding "`{`"
- code inside braces is indented two spaces
- In RStudio, use "Code > Reindent Lines"

Comments

Note that the `while` loop is the only one we really need. The other two are for convenience.

- Here's the `for` loop for iterating through a known sequence:

  ```
  for (VARIABLE in SEQUENCE) {
    EXPRESSION
  }
  ```

  and here's how to do (almost) the same thing with `while`:

  ```
  i = 1
  while (i <= length(SEQUENCE)) {
    VARIABLE = SEQUENCE[i]
    EXPRESSION
    i = i + 1
  }
  ```

  The `for` version is easier to write and read.

- Here is a `repeat` loop for running EXPRESSION one or more times:

  ```
  repeat {
    EXPRESSION
    if (CONDITION) {
     break
    }
  }
  ```

  and here is how to do (almost) the same thing with `while`:

  ```
  EXPRESSION
  while (!CONDITION) {
    EXPRESSION
  }
  ```

  The `repeat` version is better because having two copies of `EXPRESSION` is hard to maintain.

Example of nested loops

Recall matrix multiplication, in which $C_{n\times p} = A_{n\times m}B_{m\times p}$, where the element of C in row i and column j is $c_{i,j} = \sum_{k=1}^{m} a_{i,k} \cdot b_{k,j}$, the dot product of A's ith row and B's jth column.

R has a `%*%` operator for $C = AB$. Here is a function to do it as an example of nested loops.

```
matrix.multiply = function(A, B, debug=FALSE) {
  a = dim(A)
  n = a[1] # Number of rows of A.
  m = a[2] # Number of columns of A.
  b = dim(B)
  stopifnot(m == b[1]) # Otherwise A and B cannot be multiplied.
  p = b[2] # Number of columns of B.

  C = matrix(data=rep(x=0, times=n*p), nrow=n, ncol=p) # n by p zeros.
  for (i in 1:n) { # For each row in C.
    if (debug) {
      cat(sep="", "i=", i, "\n") # for debugging
    }
    for (j in 1:p) { # For each column in C.
      if (debug) {
        cat(sep="", "  j=", j, "\n") # for debugging
      }
      # Set C[i, j] to dot product of ith row of A and jth column of B.
      for (k in 1:m) {
        C[i, j] = C[i, j] + A[i, k] * B[k, j]
        if (debug) {
          cat(sep="", "    C[", i, ", " , j, "]=", C[i, j], "\n") # for debugging
        }
      }
    }
  }
  return(C)
}

A = matrix(data=1:6, nrow=2, ncol=3)
B = matrix(data=1:6, nrow=3, ncol=2)
A
B
matrix.multiply(A, B)
A %*% B # Does our function give same results as R's operator?

matrix.multiply(A, B, debug=TRUE) # Note debugging output.
```