`for` loops

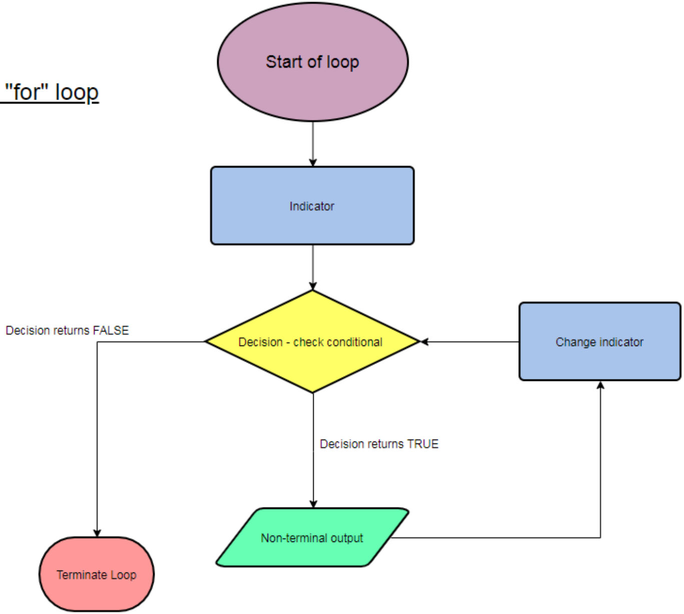
Stephen Mullins

7/16/2019

The Concept of `for` Loops

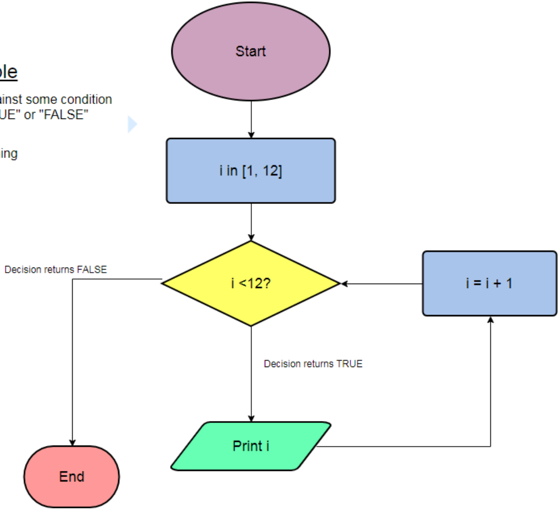
- Recall how we used “rnorm” x to produce a bunch of random values in our vector `random x`?
- Populating a vector, matrix, or higher-dimensional array in that way is based on iterative assignment of values
- We’re going to do this with a `for` loop

Basic "for" loop



Basic "for" loop - example

- Creates an indicator (i) to be checked against some condition
    - The check will return a value of "TRUE" or "FALSE"
- If the condition is:
    - TRUE: the loop tells R to do something
    - FALSE: the loop stops



The Concept of `for` Loops - Example 1

```r
for(i in 1:12){ #Start the loop
  print(i)  #Non-terminal output
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 11
## [1] 12
```

The Concept of `for` Loops - Example 1

- R automatically handles checking `i` against the range given
- R also iterates the loop as long as the conditional returns TRUE

The Concept of for Loops - Example 2

```r
newrandom=0

for(i in 1:5){
  newrandom[i]=rnorm(x, n=1)
  print(newrandom)
}
```

```
## [1] -2.835783
## [1] -2.835783 -5.384838
## [1] -2.835783 -5.384838 -2.790444
## [1] -2.835783 -5.384838 -2.790444 -4.315000
## [1] -2.835783 -5.384838 -2.790444 -4.315000 -5.294611
```

for Loops - What's the output?

```
x2=c(1, 2, 3, 4, 5, 6)

for(i in x2){
  if(i %% 2){
    print(i)
  }
}
```

for Loops - What's the output?

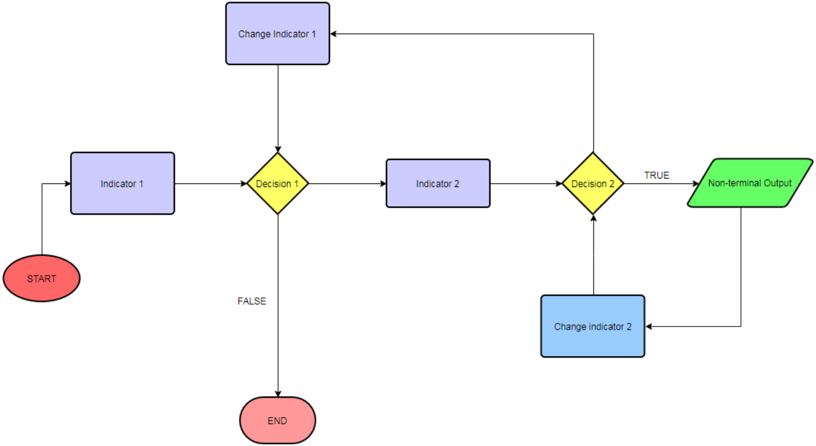
```r
x2=c(1, 2, 3, 4, 5, 6)

for(i in x2){
  if(i %% 2){
    print(i)
  }
}
```

```
## [1] 1
## [1] 3
## [1] 5
```

for Loops - Nesting

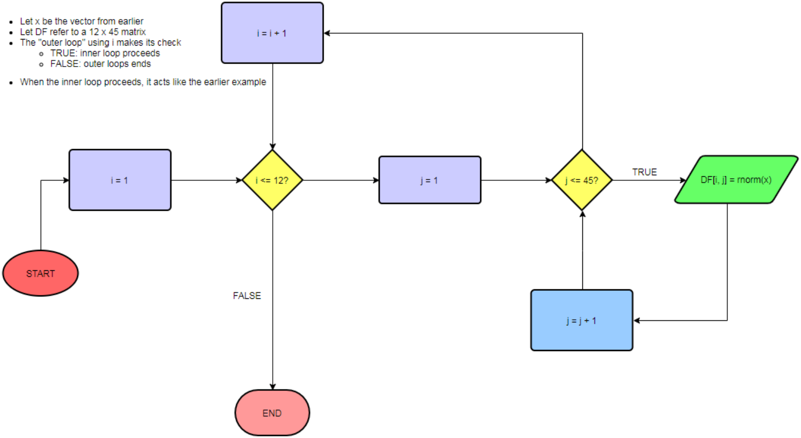
- Loops, just like “if-else” statements, can be nested
- Nested `for` loops can allow us to iterate over arbitrarily many dimensions of an array
- They can also allow us to embed more complicated processes into any given stage of the output

Nested "for" loop



Nested "for" loop

- Let x be the vector from earlier
- Let DF refer to a 12 x 45 matrix
- The "outer loop" using i makes its check
    - TRUE: inner loop proceeds
    - FALSE: outer loops ends
- When the inner loop proceeds, it acts like the earlier example



for Loops - Nested Example

```
newrandom=matrix(0L, nrow=3, ncol=3)

for(i in 1:3){
  for(j in 1:3){
    newrandom[i,j]=rnorm(x, n=1)
  }
}
```

`for` Loops - Nested Example

```
print(newrandom)
```

```
##           [,1]      [,2]      [,3]
## [1,] -3.221318 -3.854555 -4.524445
## [2,] -4.809018 -2.212966 -3.332962
## [3,] -4.293674 -2.595778 -6.392402
```

`for` Loops - Nested Output?

```
newrandom=matrix(0L, nrow=3, ncol=3)

for(i in 1:3){
  for(j in 1:3){
    newrandom[i, j]=sample(x2, size=1)
  }
}
```

for Loops - Nested Output?

```
print(newrandom)
```

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

for Loops - Further Nesting

- You can also nest “if-else” and other statements/functions inside a `for` loop
- Why might you want to do that?

for Loops - Nested “if-else”

```
newrandom=matrix(0L, nrow=3, ncol=3)

for(i in 1:3){
  for(j in 1:3){
    a=sample(x2, size=1)
    if(a %% 2){
      newrandom[i,j]=a
    }
    }
  }
```

for Loops - Nested “if-else”

```r
print(newrandom)
```

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

for Loops - Control Flow

- Another important aspect of writing loops is control flow
- Good control flow allows you to manipulate how nested loops run

`for` Loops - Control Flow

- `break` and `next` are two useful statements for control flow in loops
- `break` terminates the current loop and moves “up” a level
- `next` skips the current iteration *without* terminating the current loop

Using break and next for Control Flow

```
newrandom=matrix(0L, nrow=3, ncol=3)

for(i in 1:3){
  for(j in 1:3){
    a=sample(x2, size=1)
    if(mean(newrandom[i]>4)){
      break
    }else{
      newrandom[i,j]=a
    }
  }
}
```

Using `break` and `next` for Control Flow

```r
print(newrandom)
```

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

Using break and next for Control Flow

```r
newrandom=matrix(0L, nrow=3, ncol=3)

for(i in 1:3){
  for(j in 1:3){
    a=sample(x2, size=1)
    if(a>=5){
      next
    }else{
      newrandom[i,j]=a
    }
  }
}
```

Using `break` and `next` for Control Flow

```r
print(newrandom)
```

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

repeat and stop

- repeat sets up a loop in which break must be called explicitly *in* the loop in order to terminate it. This can allow you to declare arbitrarily many mutually-exclusive break conditions, and can be useful if you need to nest multiple logical checks within a single larger loop.
- stop tells R to cease evaluating code and, optionally, produce an error message. If you want *everything* to terminate in some undesirable fringe case, this is useful.

`for` Loops - Your turn!

- To develop a stronger working understanding of `for` loops, you will now complete some practice problems
- You are encouraged to collaborate, and to ask questions of the instructors if you need assistance

Problem 1

- a) Write pseudocode giving the structure of a `for` loop that prints the values of a list of numbers `g = {1, 3, 7, 4, 6, 3, 2, 2}`
- b) After checking with your instructor, write this loop in Rstudio and print the output to the console.
- c) Using 1a and 1b as a basis, write a loop that outputs `g*2` by multiplying element-wise

Problem 2

- a) We want unique results. Repeat Problem 1a, but add an `if` statement that discards the duplicate 2.
- b) Check your pseudocode and write the loop with nested condition.
- c) Repeat 1c, but now produce `g*2` for all *odd* elements and `g/2` for all *even* elements

Problem 3

- Construct a 6x6 matrix named `gmat` populated with zeroes.
- a) Using a nested `for` loop, populate each location with a random number from `g`. Check your pseudocode.
- b) Reset `gmat`. Use a nested `for` loop to populate *only* the main diagonal with random numbers from `g`.

Problem 4

- a) Reset `gmat`. Use a nested `for` loop to populate *only* the off-diagonal elements. Skip the second position using `next` if a number is drawn twice in a row (e.g. if `gmat[1,2]`=3, `gmat[1,3]` $\neq$ 3).
- b) Reset `gmat`. Use a nested `for` loop with a(n) `if` statement and `break` to skip the rest of the row if a number is drawn twice in a row (e.g. if `gmat[1,2]`=3, `gmat[1,3:6]` should not be populated if `gmat[1,3]`=3).

Problem 5

- Reset `gmat`. Use what we have learned to populate only the off-diagonal elements of `gmat` with random numbers from `g`. If the sum of the matrix indices `i` and `j` is even for a position, only populate that position with an even number from `g`. If the sum of the matrix indices is odd, the element at that position should be odd. If the same number is drawn twice in a row for a position, skip that position. If the same number is drawn more than 18 times in the process of populating the `gmat`, terminate the entire operation.

Problem 5 - Choosing Code

- To tie it all together, we will now do the following:
- Reset `gmat`. Populate only the off-diagonals with random numbers from `g`. We will use only even numbers for elements of `gmat` where `i` and `j` sum to an even number, and only odd numbers for positions whose index sum is odd. We will also skip positions entirely if we draw the same number from `g` twice in a row. If we happen to sample the same number from `g` more than 18 times in the entire process, we will terminate the loop.