Lecture 11: Sept 26, 2018 - v2

Loop-the-Loops

- *Iteration*
  - *while*, *for*, and *repeat*
  - *next*, *break*

James Balamuta
STAT 385 @ UIUC

CC BY NC SA

Announcements

- **hw04** is due **Friday, Sep 28th, 2018** at **6:00 PM**
- **Quiz 05** covers Week 4 contents @ CBTF.
  - Window: Sep 25th - 27th
  - Sign up: https://cbtf.engr.illinois.edu/sched
- Want to review your homework or quiz grades? **Schedule an appointment.**
- Got caught using GitHub's web interface in hw01 or hw02? Let's chat.

Last Time

- **Testing Frameworks**
  - Steps of a Hypothesis Test
  - Sampling distributions play a large role.
  - Case study in implementing a method for assessing group difference.

Lecture Objectives

- **Explain** the difference between iteration structures.
- **Understand** the pros and cons of each iteration structure.
- **Decide and implement** the best iteration structure for a given scenarios.

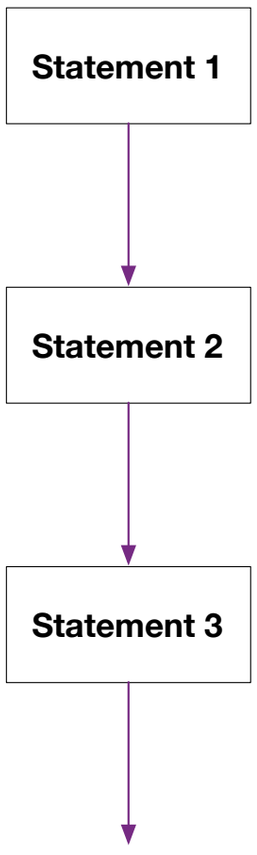
Iteration

Previously

Kinds of Structures

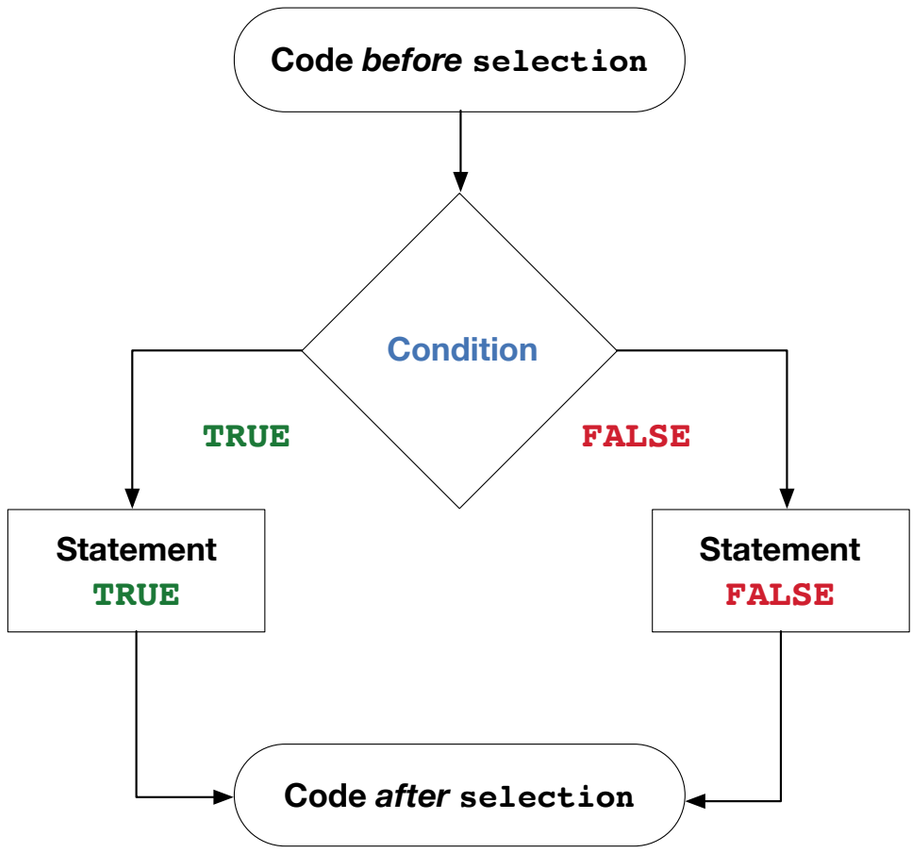
… how computers process information …

Sequential



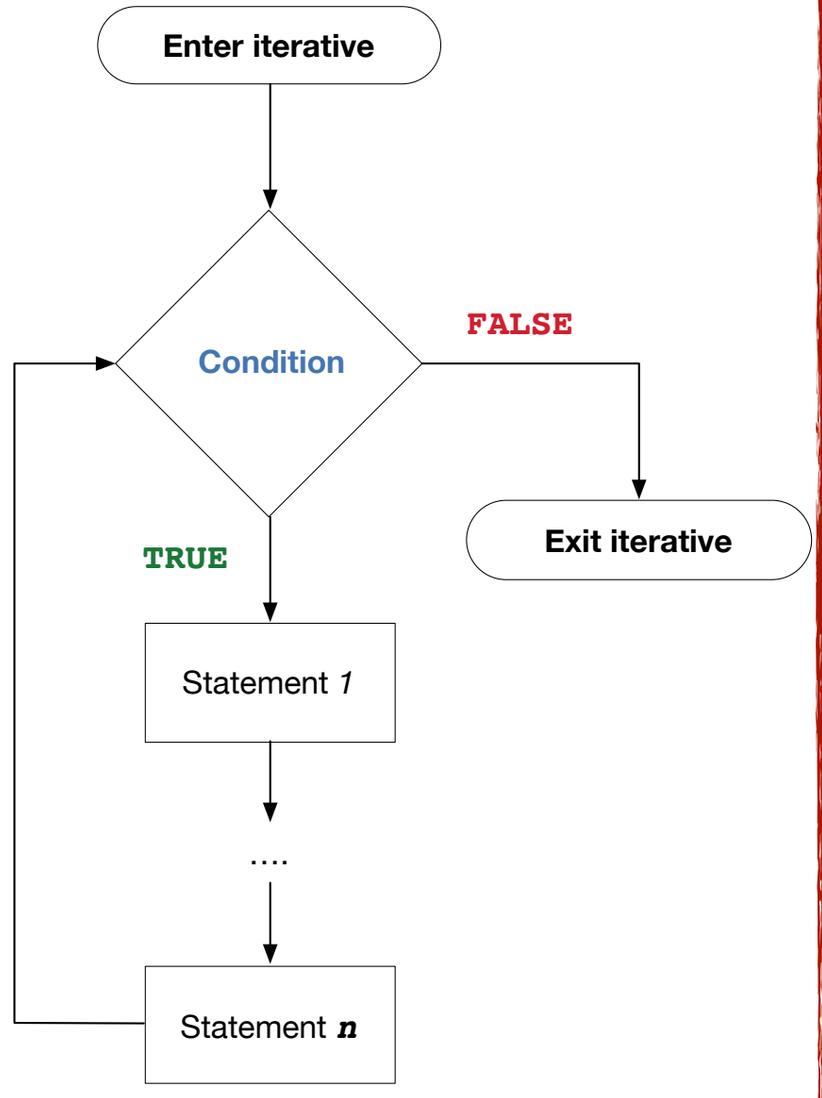
"One foot in front of the other"

Selection



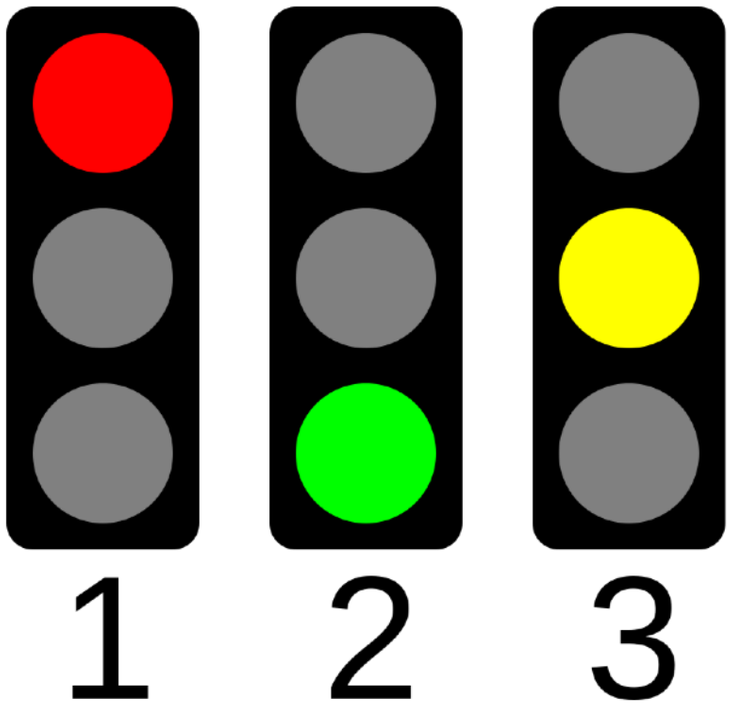
"Making a choice"

Iterative

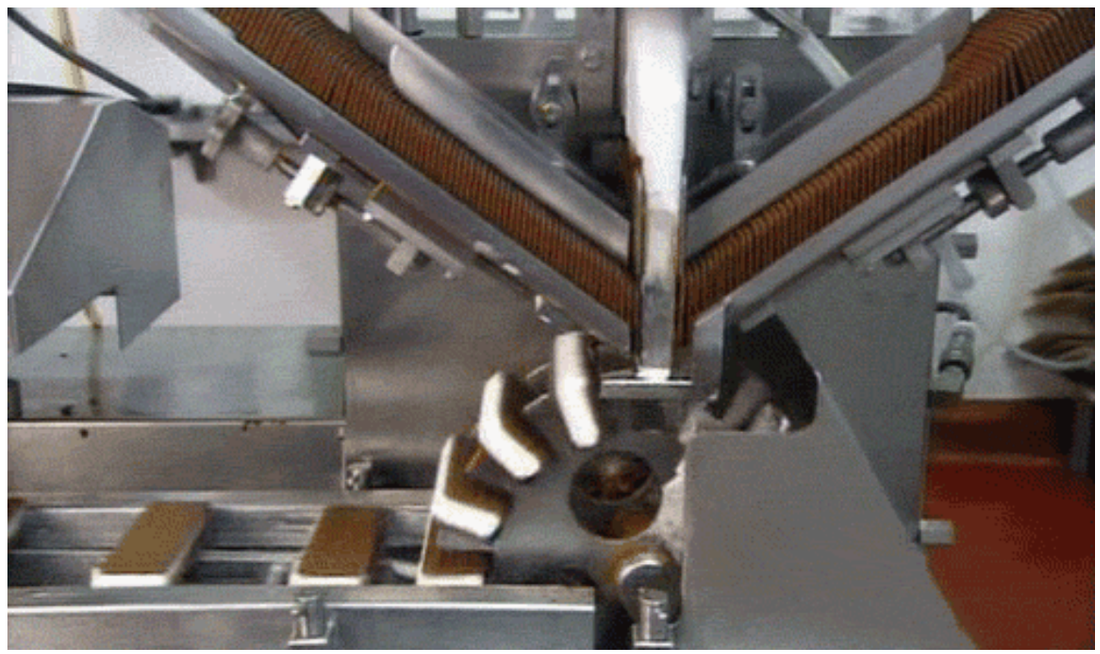


"Repeating yourself"

Definition:
Iteration is a computational structure that allows the computer to repeat the same instruction multiple times.



Source



Source

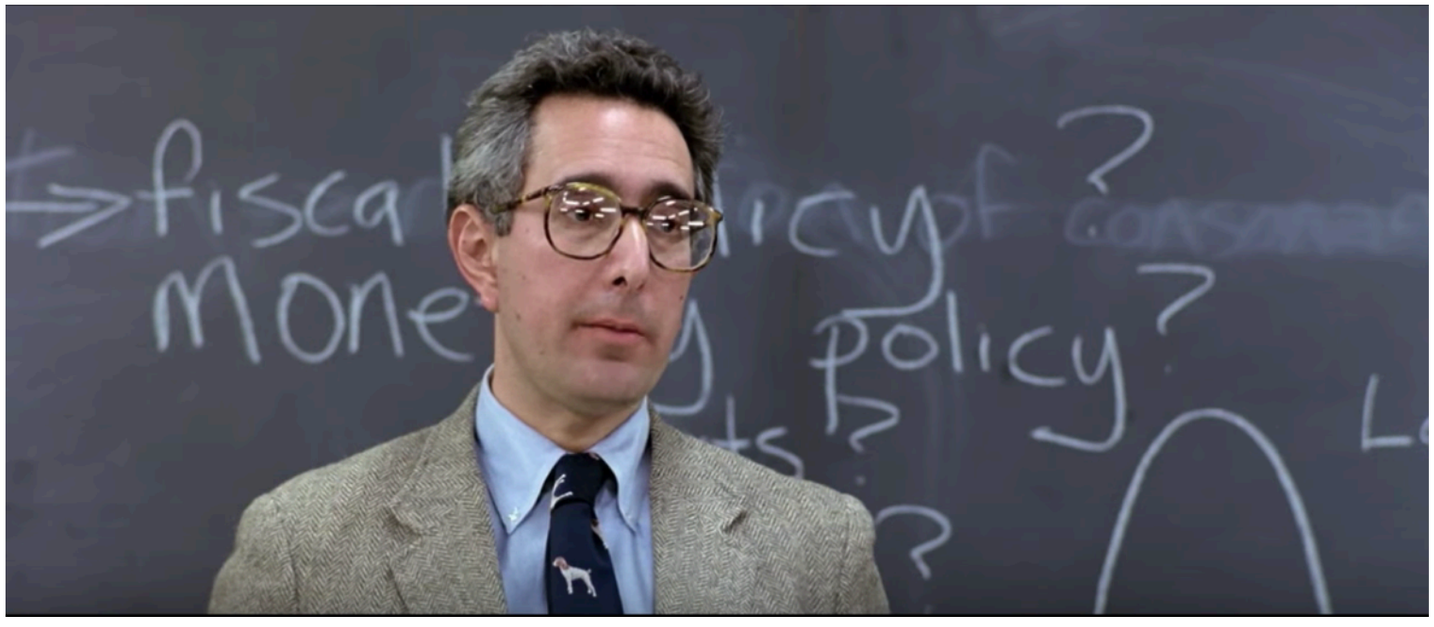
Computers are great with repetition due to

strict adherence of instructions

Humans... Not so much.

Quality of Repetition

… how humans talk …



Bueller, Bueller, Bueller

vs.

Anyone? Anyone? Anyone?

Anyone?

Anyone? This is…

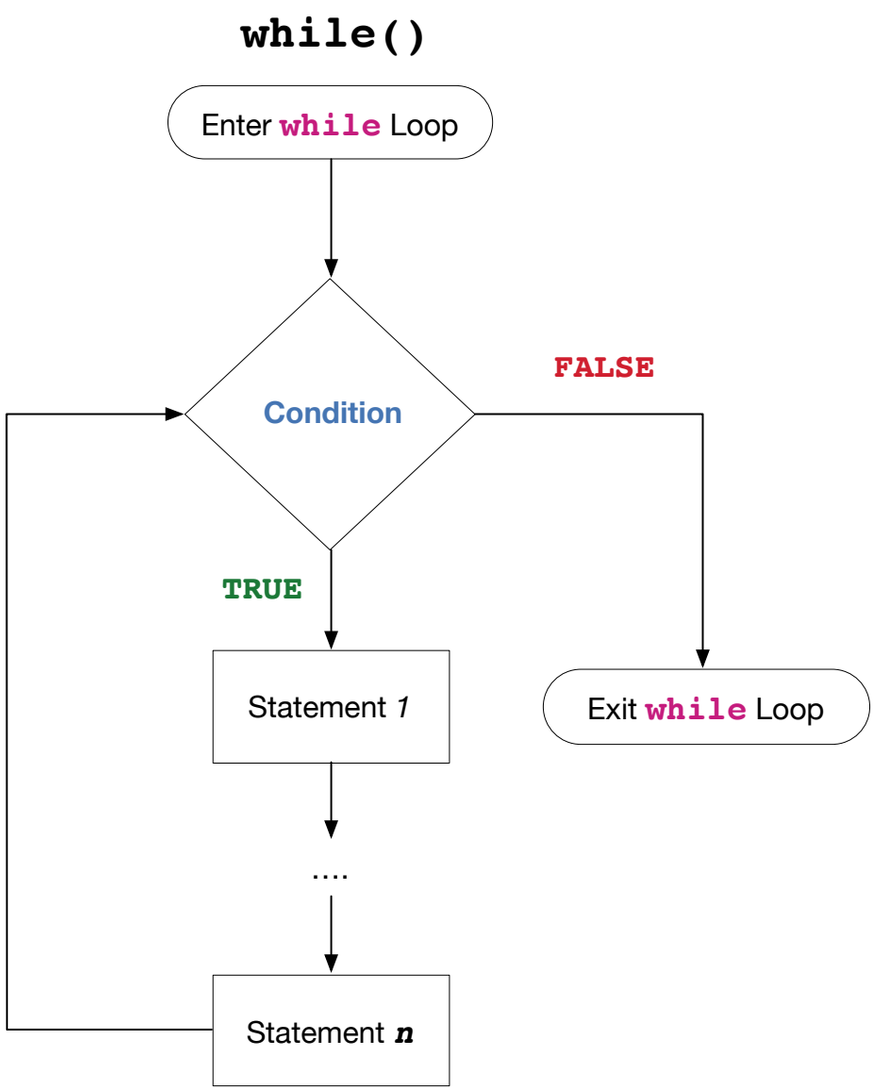
Overview of Iteration in *R*

… element access types …

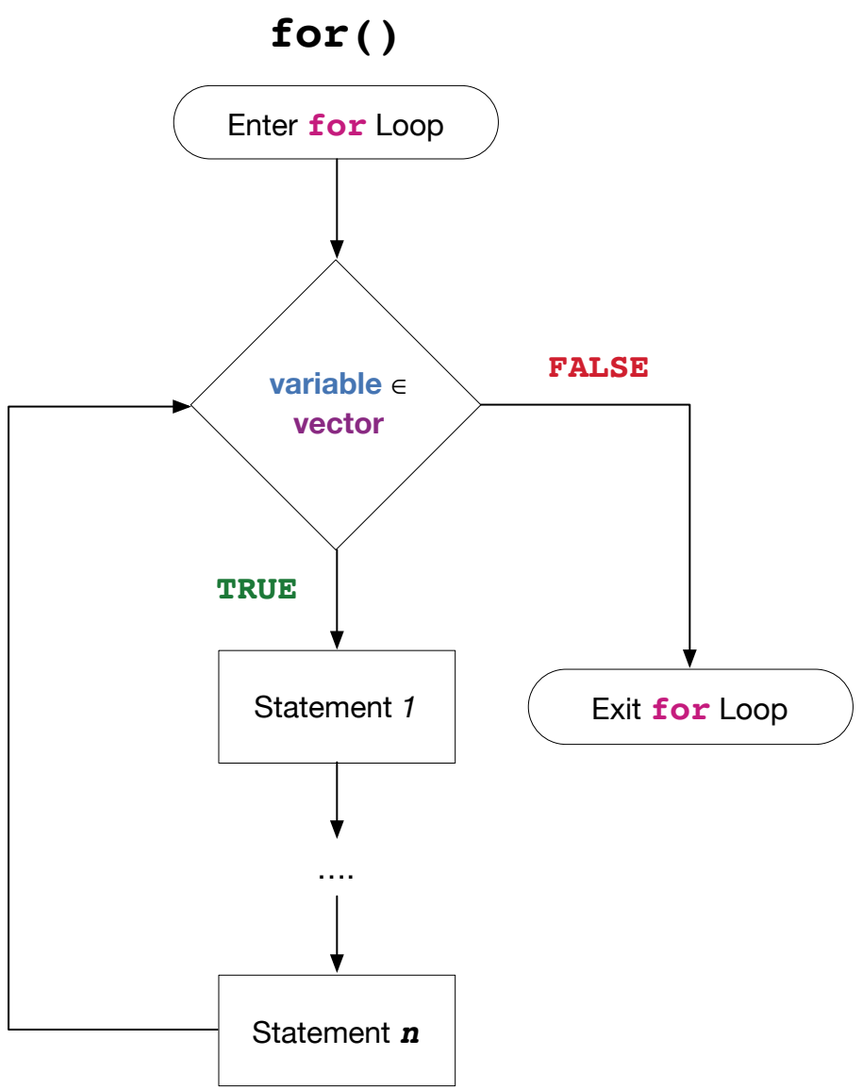
- *Implicit* Iteration
  - **Vectorization**
    - covered previously
  - **Functionals**
    - future, c.f. the ***apply()** family and **purrr::map*()**
- *Explicit* Iteration
  - **for**(), **while**(), **repeat**{}

Iteration Structures

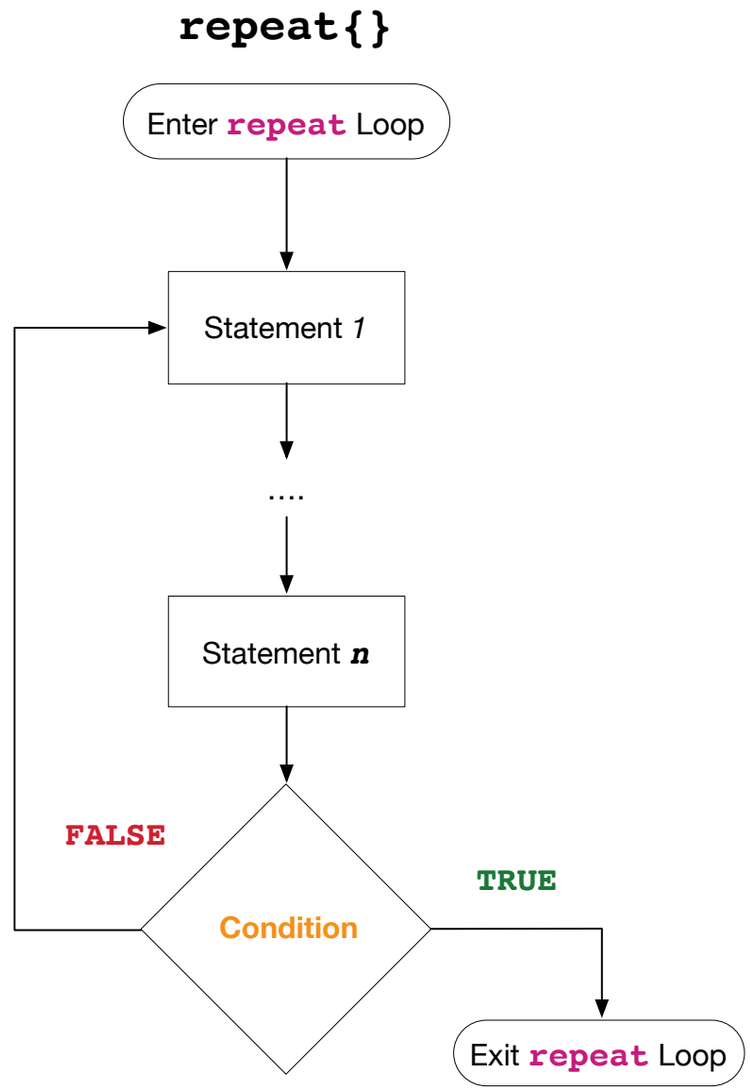
… an overview of how to repeat or loop information …



- Stopping condition known only after starting
- More versatile **for** loop.



- Known amount of repeats
- Index or character access required



- First iteration should always occur.
- Equivalent to **while(TRUE)**

while loop template

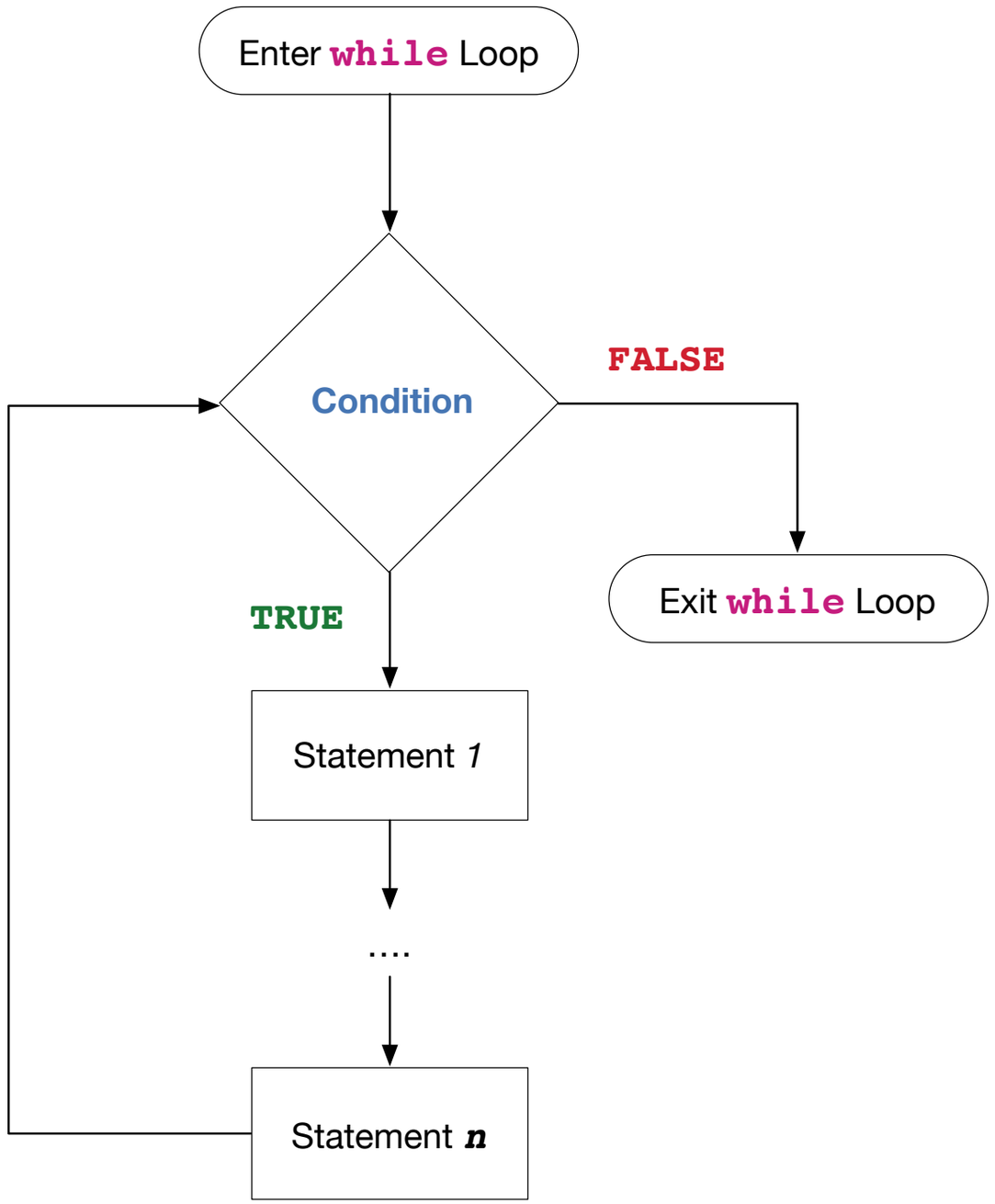
… only run if condition is true …

Iteration Structure
Declaration of the iteration structure to use when perform repeat calculations

Logical Condition
Test to see whether the loop should continue (TRUE) or stop (FALSE)

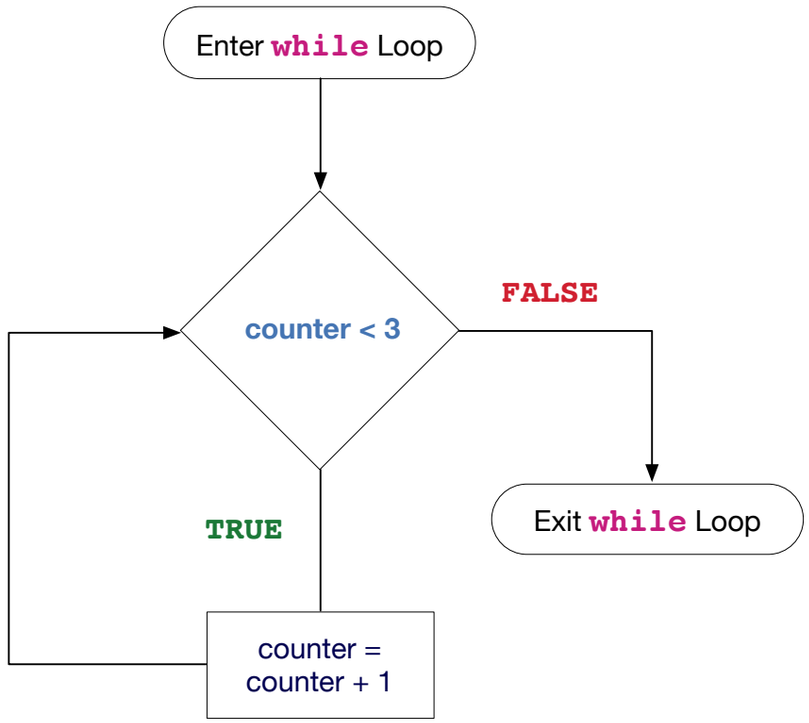
```
while (condition) {
    Statement 1
    .....
    Statement n
}
```

Body
Statements in between {} are run on each iteration of the loop



Counting up to **3**

… why not **4** ???



counter_start	Logical Condition	counter_end
0	TRUE	1
1	TRUE	2
2	TRUE	3
3	**FALSE**	**3**

Initialization Statements
Statements used inside the *iteration* structure that must be defined prior to use

```
counter = 0
```

Iteration Structure
Declaration of the iteration structure to use when perform repeat calculations

Logical Condition
Test to see whether the loop should continue (TRUE) or stop (FALSE)

```
while (counter < 3) {
    counter = counter + 1
}
```

Body
Statements in between {} are run on each iteration of the loop

Your Turn

Write a **while** loop that counts down from **10** to **1** and then says **"Blast off!"**

```
_____ = _____
while ( _____ ) {
    _____
    _____ = _____ ___ _____
}
_____
```

Gambler's Ruin

… avoiding going broke …

```
money = 5
set.seed(1115)
while (money > 0 && money =< 10) {
    message("I have ", money , "$! Let's gamble!")
    coin_flip_result = rbinom(1, 1, 0.4)
    if (coin_flip_result == 1) {
        money = money + 1
    } else {
        money = money - 1
    }
}
message("I walked away with ", money , "$!!")
```

A Loop to Infinity

… when a loop goes Coo-Coo …

```
# What happens when we remove the lower bound of the conditions
# to gamble and rig the game?

money = 5                    # Initial amount of money to gamble
while (money =< 10) { # The bank doesn't mind loaning money…

    message("I have ", money , "$! Let's gamble!")
    money = money - 1 # A gambler with bad luck always loses…

}
message("I walked away with ", money , "$!!")
```

```
# The loop failed because of a never-ending condition...
# Simplifying the prior code block would be give...

while (TRUE) {
    # Continually running the block of statements
    # inside the { }
}

# Outside of the loop is never reached.
```

Never-ending Loops

... when a loop never reaches its termination condition ...

Geometric Series

... analytical computing a solution ...

$$\sum_{k=0}^{\infty} ar^k = a + ar + ar^2 + \cdots = \frac{a}{1-r}, \quad |r| < 1$$

```
# Values of the geometric series
a = 2
r = 0.5

# Compute sum of series with formula
a/(1 - r)
# [1] 4
```

How could we **compute the series sum** if we **didn't know** the analytical formula?

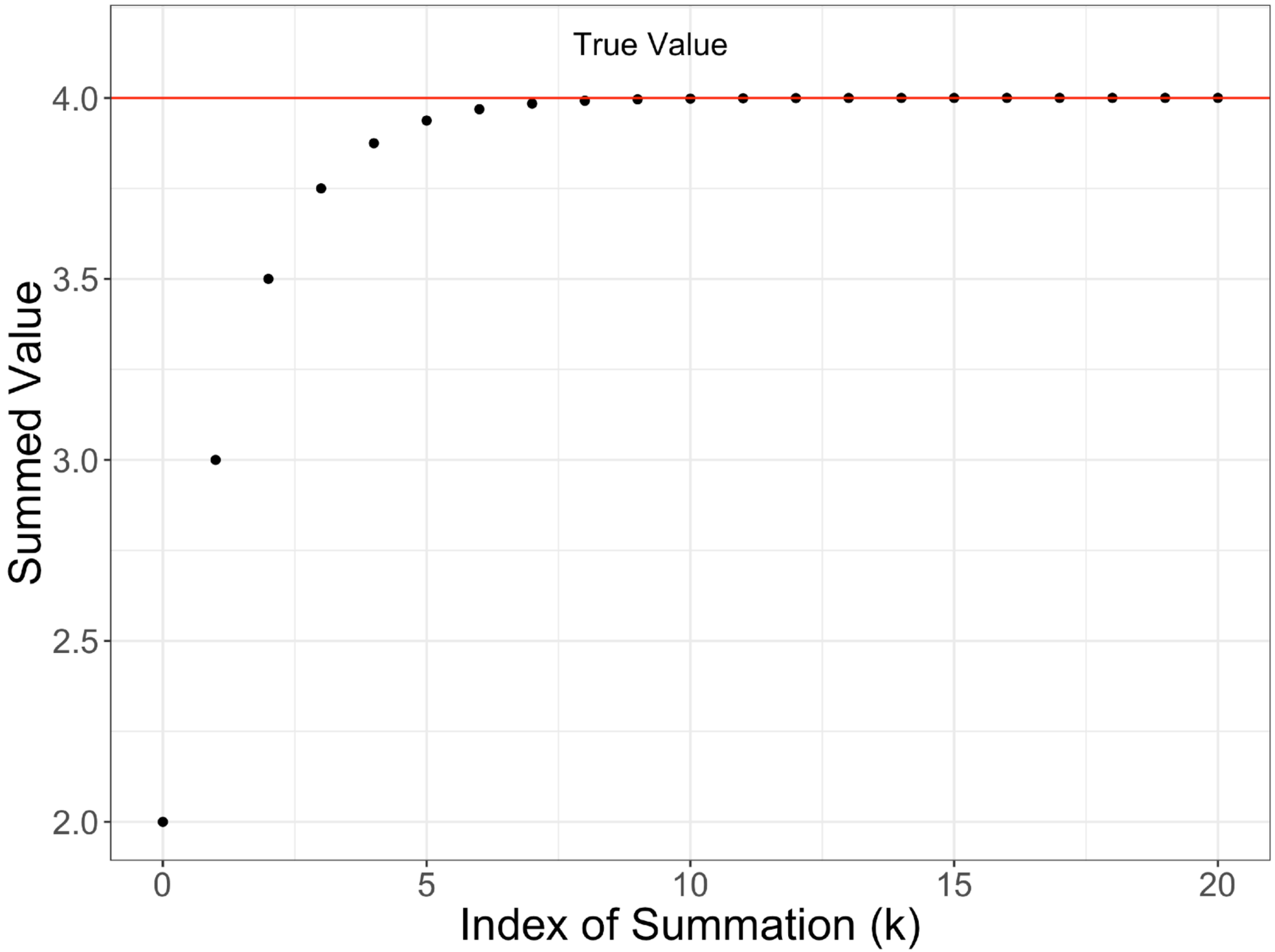
Calculating Summation Terms

… individually calculating and then adding terms …

```
# Compute each term by itself
round(a * r^(0:20), 4)
# [1] 2.0000 1.0000 0.5000 0.2500 0.1250 0.0625 0.0312
# [8] 0.0156 0.0078 0.0039 0.0020 0.0010 0.0005 0.0002
# [15] 0.0001 0.0001 0.0000 0.0000 0.0000 0.0000 0.0000
# Consecutive terms are now somewhat equivalent…

# Cumulative sum will sum up all the terms up to the present point
round( cumsum(a * r^(0:20)), 4)
# [1] 2.0000 3.0000 3.5000 3.7500 3.8750 3.9375 3.9688
# [8] 3.9844 3.9922 3.9961 3.9980 3.9990 3.9995 3.9998
# [15] 3.9999 3.9999 4.0000 4.0000 4.0000 4.0000 4.0000
```

Visualizing Convergence

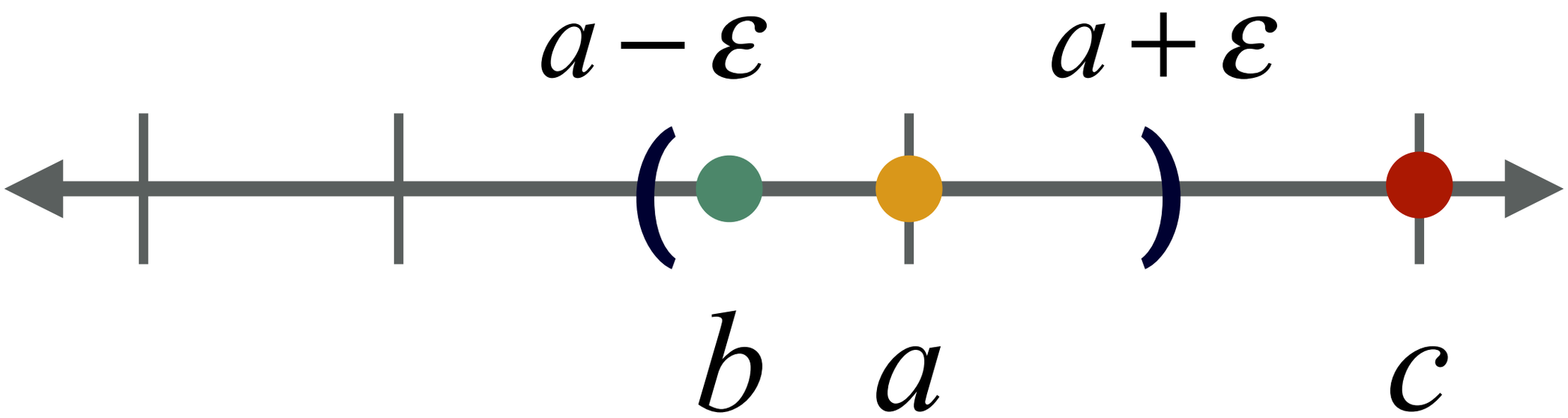
… values converging to a specific value …



Previously

Epsilon (ε) Neighborhoods

… where close enough is spot on …



- Since ***b*** is inside the epsilon neighborhood of ***a*** they are **equal.**
- With ***c*** *outside* of ***a***'s epsilon neighborhood, the values are **not equal.**

Sequence Convergence

… Approximating a Sequence …

Adding terms from $\sum_{k=0}^{\infty} ar^k$ while $\left|ar^k - ar^{k-1}\right| \geq \varepsilon$

```
eps = 0.001                    # Set a discrepancy
counter = 1                    # Keep track of number of iterations

x1 = a; x2 = a * r             # Compute first two terms separately
summed = x1 + x2               # Sum the terms

while( abs(x1 - x2) >= eps) { # Any difference between terms?
  counter = counter + 1        # Count loop iteration
  x1 = x2                      # Set last computed term to x1
  x2 = a*r^counter             # Compute new term
  summed = summed + x2         # Add new term to summation
}
summed; counter                # Display summation and n iterations
```

What if we just want to **access values in an existing vector?**

Previously

Positional Indexes

… working with positional indexes …

```
ex_vec = c(5, 3, -2, 42)
```

ex_vec

5	[1]
3	[2]
-2	[3]
42	[4]

Data

Indices (Hidden)

Summing Values

… individually adding values together from a vector …

$$\sum_{i=1}^{n} x_i = x_1 + x_2 + \cdots + x_{n-1} + x_n$$

```
values = c(-4, 2, 5, 9, 0)        # Values to add together
summed = 0                        # Summed value
counter = 1                       # Keep track of number of iterations

while(counter < length(values)) {             # Any elements left?
   summed = summed + values[counter]          # Add value onto summed
   counter = counter + 1                      # Increment position in vector
}

summed                            # Display summation and n iterations
# [1] 12
sum(values)                       # Check output against built-in vectorized function
# [1] 12
```

Iterating until all elements are accessed is a

Common Iteration Pattern

provided by the **for** structure

for template

… known quantity iteration …

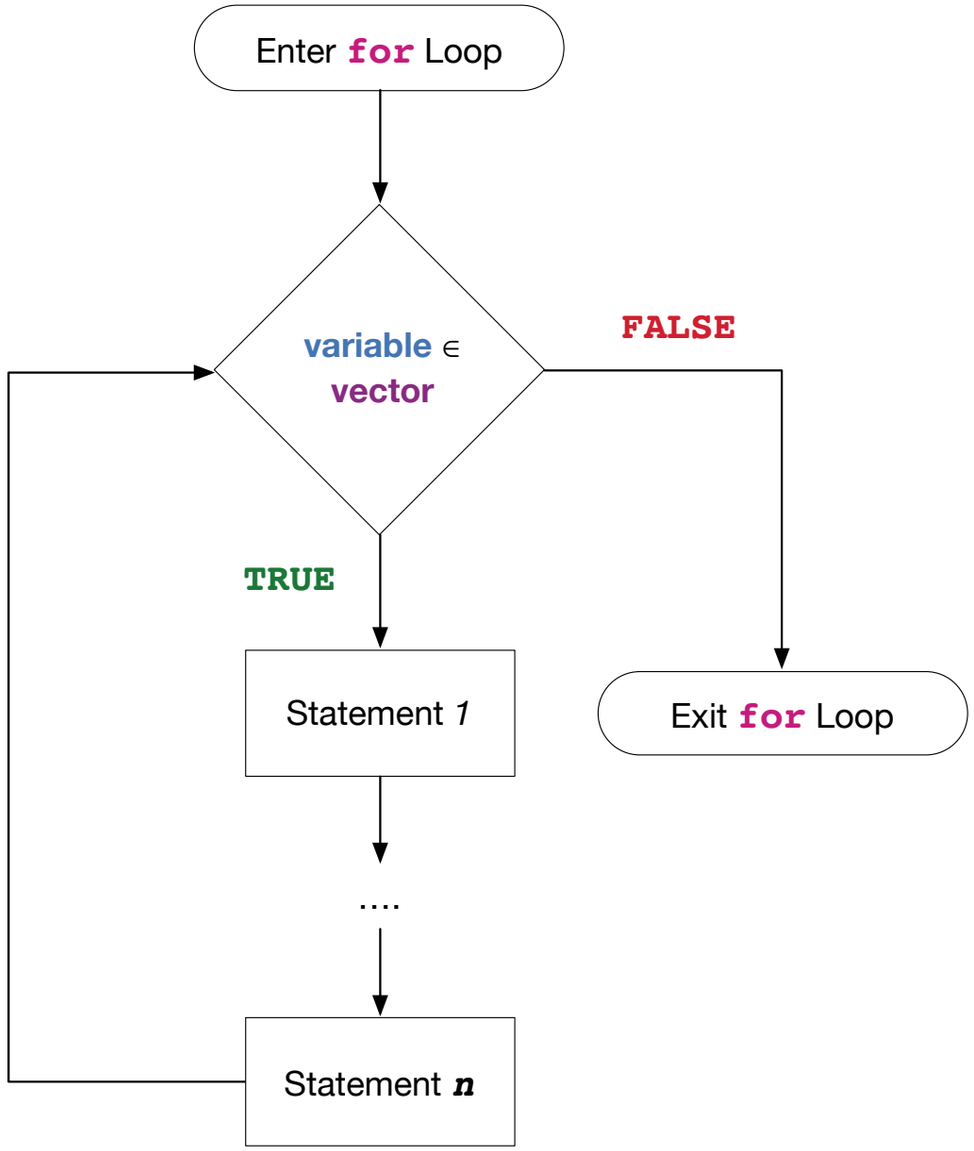
Iteration Structure
Declaration of the iteration structure to use when perform repeat calculations

Variable
Contains a value extracted from vector being operated on in one iteration of the loop

Vector
A collection of values that will be iterated over in the body statements

```
for (variable in vector) {
    Statement 1
    .....
    Statement n
}
```

Body
Statements in between {} are run on each iteration of the loop



Previously

Generating Sequences

… types of sequences …

```
ex_vec = c(5, 3, -2, 42)
1:length(ex_vec)
# [1] 1 2 3 4
seq(1, length(ex_vec))
# [1] 1 2 3 4
seq_len(length(ex_vec))
# [1] 1 2 3 4
seq_along(ex_vec)
# [1] 1 2 3 4
```

Initialization Statements
Statements used inside the *iteration* structure that must be defined prior to use

```
x = c(5, 3, -2, 42)
summed = 0
```

Iteration Structure
Declaration of the iteration structure to use when perform repeat calculations

Variable
Contains a value extracted from vector being operated on in one iteration of the loop

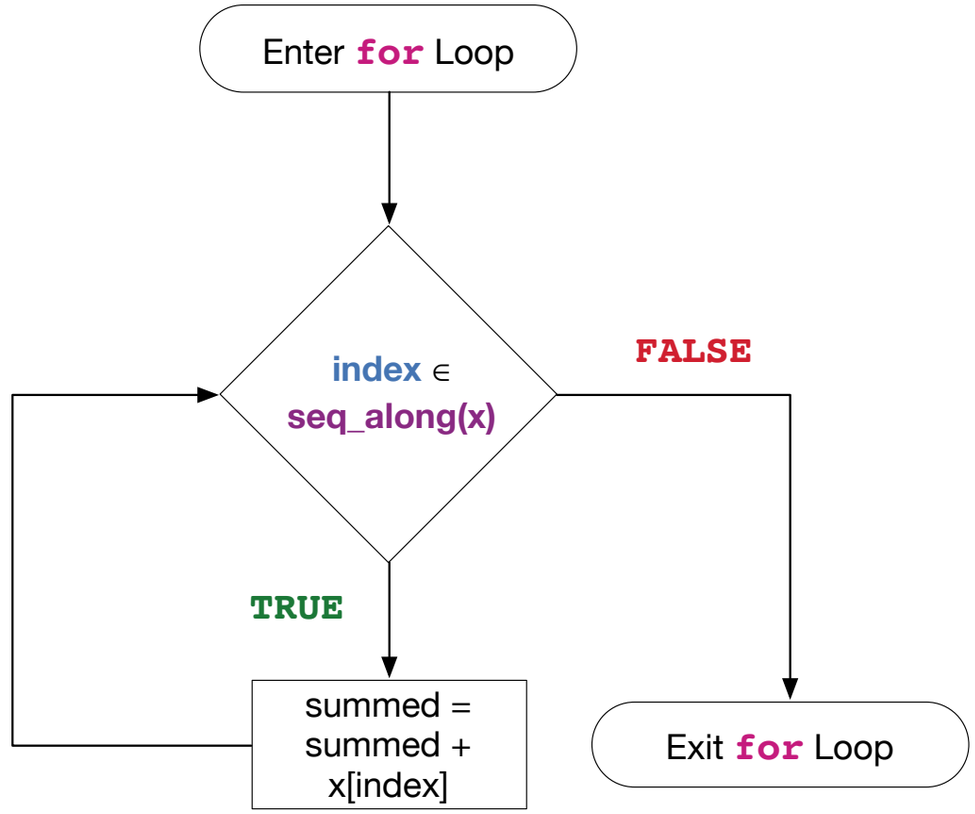
Vector
A collection of values that will be iterated over in the body statements

```
for (index in seq_along(x)) {
  summed = summed + x[index]
}
```

Body
Statements in between {} are run on each iteration of the loop

Summations

… adding together numbers by **position** …



$$\sum_{i=1}^{4} x_i = 0 + 5 + \sum_{i=2}^{4} x_i$$

$$= 5 + 3 + \sum_{i=3}^{4} x_i$$

$$= 8 + -2 + \sum_{i=4}^{4} x_i$$

$$= 6 + 42 = \boxed{48}$$

Flow of Iteration

… seeing changes over time …

index	x[index]	summmed_start	summed_end
1	5	0	5
2	3	5	8
3	-2	8	6
4	42	6	**48**

1. **index**
   - the current location of the iteration;
2. **x[index]**
   - the value being accessed;
3. **VAR_start**
   - the contents of the variable *before* the modification; and
4. **VAR_end**
   - the contents of the variable *after* the modification

Initialization Statements
Statements used inside the *iteration* structure that must be defined prior to use

```
x = c(5, 3, -2, 42)
summed = 0
```

Iteration Structure
Declaration of the iteration structure to use when perform repeat calculations

Variable
Contains a value extracted from vector being operated on in one iteration of the loop

Vector
A collection of values that will be iterated over in the body statements

Body
Statements in between {} are run on each iteration of the loop

```
for (index in seq_along(x)) {
  summed = summed + x[index]
}
```

for vs. **vectorization**

… why vectorization is powerful for independent cases …

Initialization Statements
Statements used inside the *iteration* structure that must be defined prior to use

```
x = c(1, 2, 3, 4)
y = numeric(length(x))
```

Iteration Structure
Declaration of the iteration structure to use when perform repeat calculations

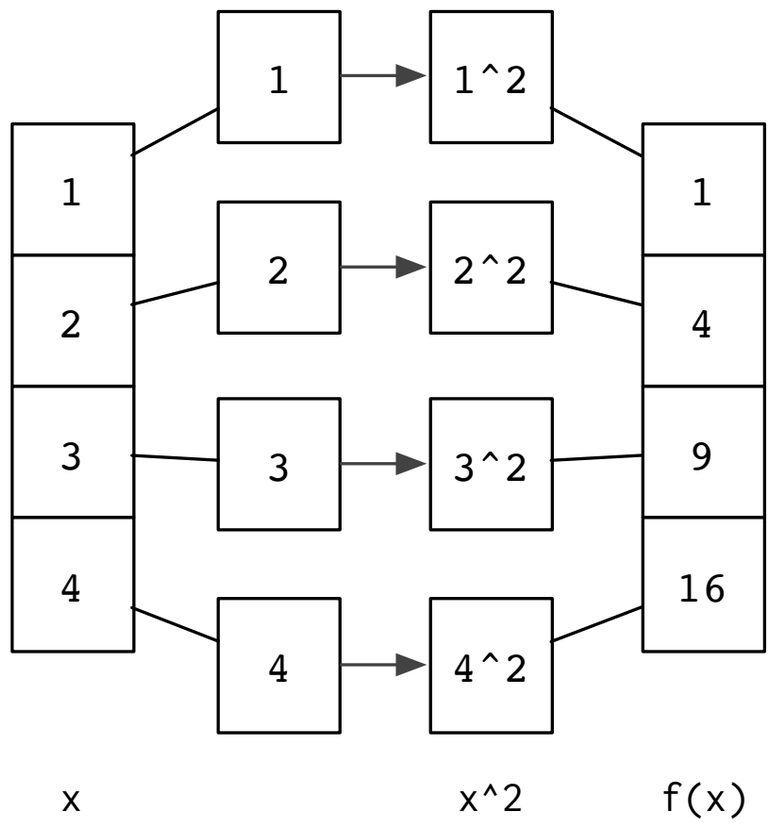
Variable
Contains a value extracted from vector being operated on in one iteration of the loop

Vector
A collection of values that will be iterated over in the body statements

```
for (index in seq_along(x)) {
  y[index] = x[index]^2
}
```

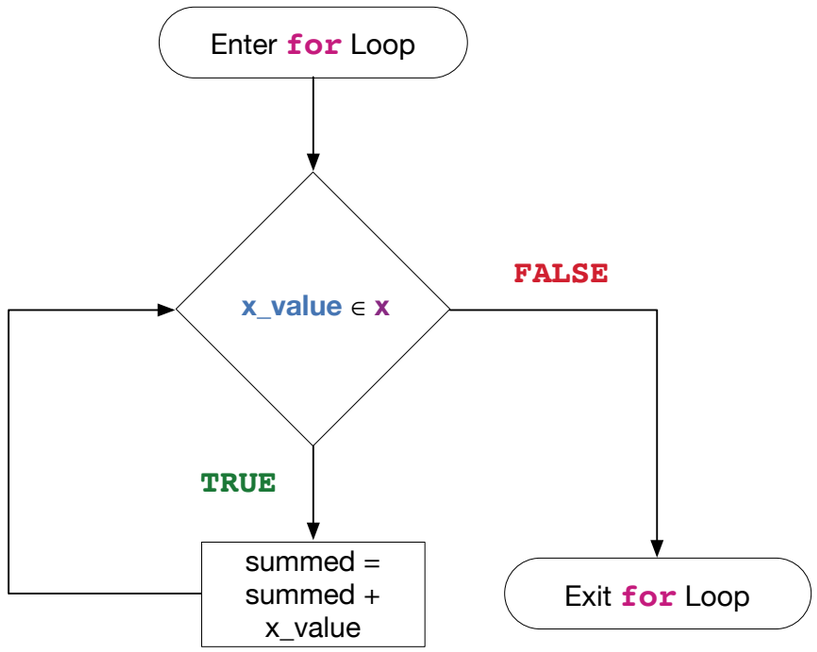
Body
Statements in between {} are run on each iteration of the loop

```
x = c(1, 2, 3, 4)
x^2
```



Redux Summations

… adding together numbers by **element** …



x_value	summmed_start	summed_end
5	0	5
3	5	8
-2	8	6
42	6	**48**

Initialization Statements
Statements used inside the *iteration* structure that must be defined prior to use

```
x = c(5, 3, -2, 42)
summed = 0
```

Iteration Structure
Declaration of the iteration structure to use when perform repeat calculations

Variable
Contains a value extracted from vector being operated on in one iteration of the loop

Vector
A collection of values that will be iterated over in the body statements

Body
Statements in between {} are run on each iteration of the loop

```
for (x_value in x) {
  summed = summed + x_value
}
```

Your Turn

Perform an element-wise addition between vectors **x** and **y** (e.g. z[i] = x[i] + y[i])

```
x = c(8L, -55L, 42L, 0L)
y = c(3L, -9L, 65L, 2L)
n_obs = length(x)
z = numeric(n_obs)

for (i in _______________) {
  ______ = ______
}

z
```

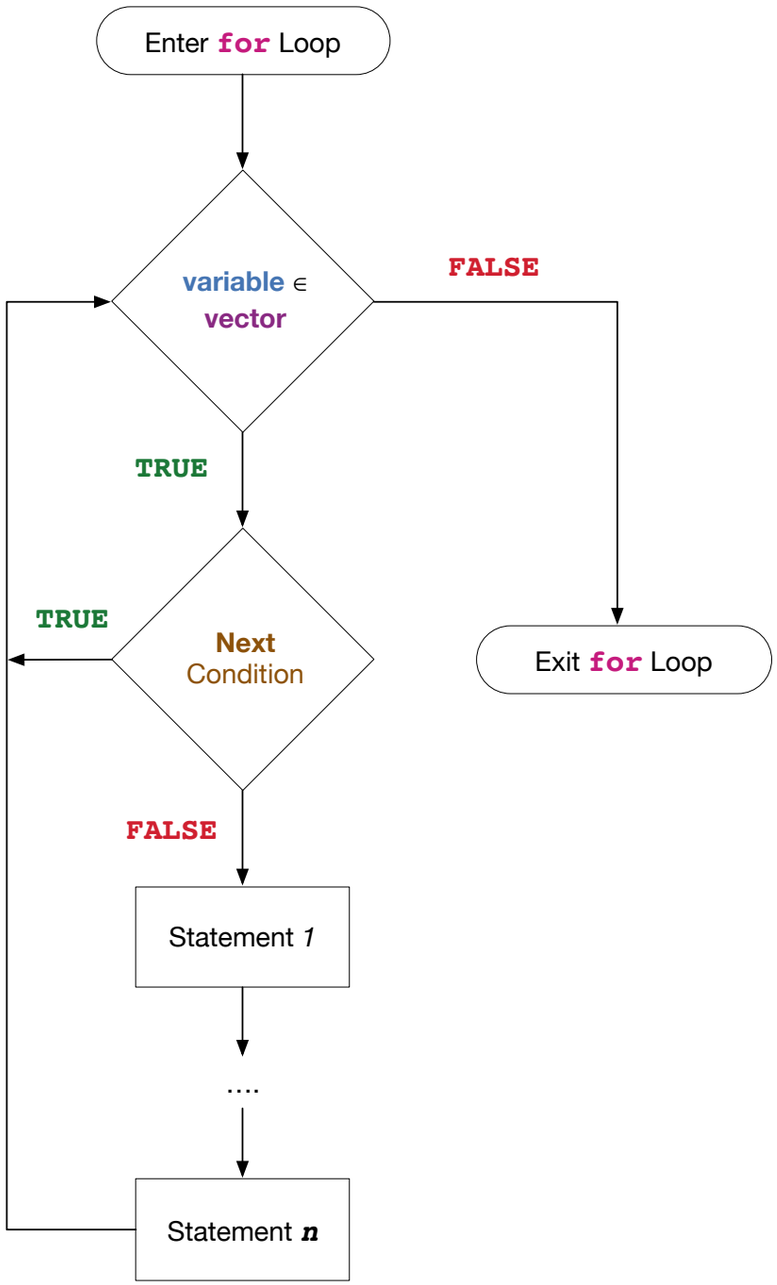
Your Turn

What will be the output of the following code? Why?

```
a = numeric()
value = 0

for (i in 1:length(a)) {
  value = value + i
}

for (i in seq_along(a)) {
  value = value - i
}

value
```

next

… skipping to the next iteration …



Iteration Structure
Declaration of the iteration structure to use when perform repeat calculations

Variable
Contains a value extracted from vector being operated on in one iteration of the loop

Vector
A collection of values that will be iterated over in the body statements

```
for (variable in vector) {
    if (<condition>) {
        next
    }
    Statement 1
    .....
    Statement n
}
```

Next Condition
Skip statements and go to the next element

Body
Statements in between {} are run on each iteration of the loop

Initialization Statements
Statements used inside the *iteration* structure that must be defined prior to use

```
x = c(5, 3, -2, 42)
summed = 0
```

Iteration Structure
Declaration of the iteration structure to use when perform repeat calculations

Variable
Contains a value extracted from vector being operated on in one iteration of the loop

Vector
A collection of values that will be iterated over in the body statements

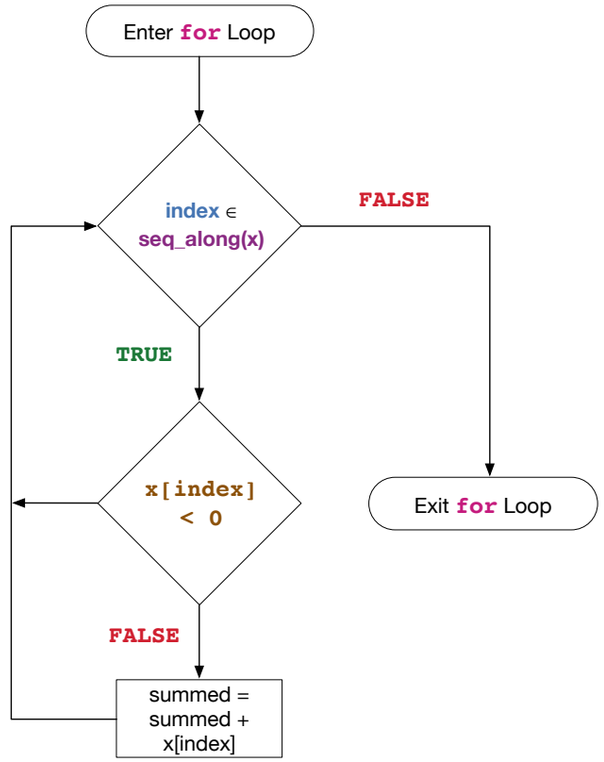
```
for (index in seq_along(x)) {
    if (x[index] < 0) {
        next
    }
    summed = summed + x[index]
}
```

Next Condition
Skip statements and go to the next element

Body
Statements in between {} are run on each iteration of the loop

Positive Numbers Sum

… adding only positive numbers …



index	summmed_start	summed_end
1	0	5
2	5	8
3	8	8
4	8	**50**

No Change

Your Turn

Add up only elements in **a** that do not have a missing value.

```
a = c(-1L, -24L, NA, 11L, 0L, NA)
_____ = _____
for (i in _____________) {
   if (   ____________________________ ) {
     _____
   }
  _____ = _____ __ _____
}

_____
```

```r
# Example Iterative Classification
n_obs = nrow(bp_data)                          # Count number of observations
classify_vector = character(n_obs)             # Create an empty character vector (e.g., "")
for (i in seq_len(n_obs)) {                    # Iterate through a vector containing positional indexes
  test_obs_value = bp_data$Systolic[i]         # Retrieve ith value being tested to avoid multiple subsets in condition
  classify_vector[i] = if( test_obs_value < 120 ) {  # Classify Systolic value and save it into the vector at position i
    "Normal"
  } else if ( test_obs_value < 129 ) {
    "Elevated"
  } else if ( test_obs_value < 139 ) {
    "Stage 1"
  } else {
    "Stage 2"
  }
}
bp_data$BP_Type = classify_vector              # Add classifications to bp_data
```

Subject ID	Sex	Systolic (mm Hg)	BP Type
S005	Male	110	Normal
S130	Female	141	Stage 2
S023	Male	125	Elevated
S098	Male	168	Stage 2
S035	Male	115	Normal
S007	Female	122	Elevated
S104	Female	135	Stage 1

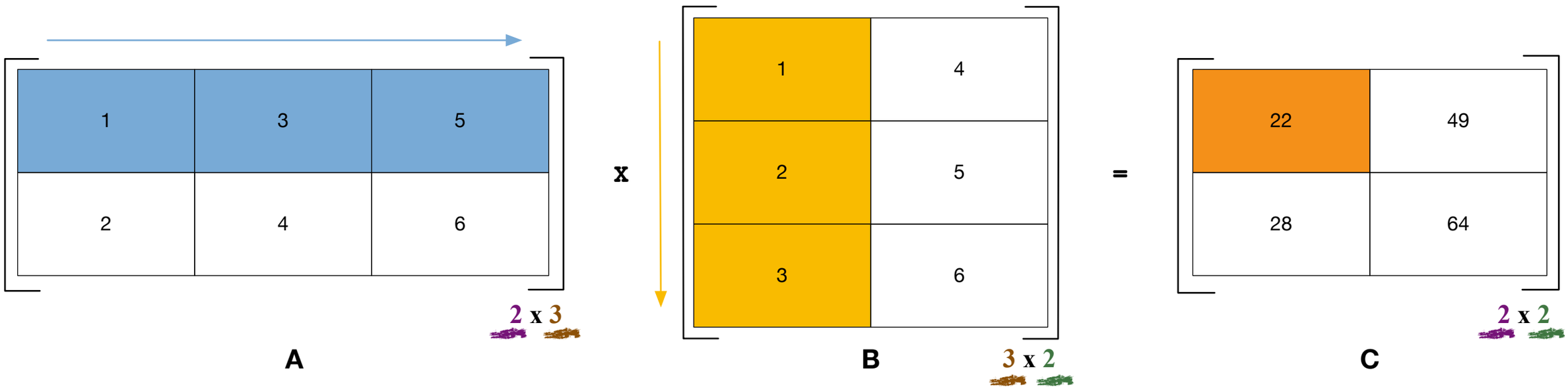
bp_data 7 x 4

Systolic	BP Type
Below 120	Normal
120-129	Elevated
130-139	Stage 1 / Hypertension
140 or higher	Stage 2 / Hypertension

Classification Table 4 x 2

Matrix Multiplication

… multiplying matrices x and y…



If **A** is an *n* x *m* matrix and **B** is an *m* x *p* matrix, then **C** = **AB** is an *n* x *p* matrix. The elements of **C** are computed with

$$c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}$$

```r
# Create a sequence
x = seq(1, 6)

# Construct matrices
A = matrix(x, nrow = 2, ncol = 3)
#.     [,1] [,2] [,3]
# [1,]    1    3    5
# [2,]    2    4    6

B = matrix(x, nrow = 3, ncol = 2)
#      [,1] [,2]
# [1,]    1    4
# [2,]    2    5
# [3,]    3    6

# Multiple Matrices
C = A %*% B
#      [,1]  [,2]
# [1,]   22   49
# [2,]   28   64
```

Example Matrix Multiplication

… multiplying two matrices …

$$A = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}_{2\times3} \quad B = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}_{3\times2}$$

$$C_{2\times2} = A_{2\times3} \cdot B_{3\times2}$$

$$C = \begin{bmatrix} 22 & 49 \\ 28 & 64 \end{bmatrix}_{2\times2}$$

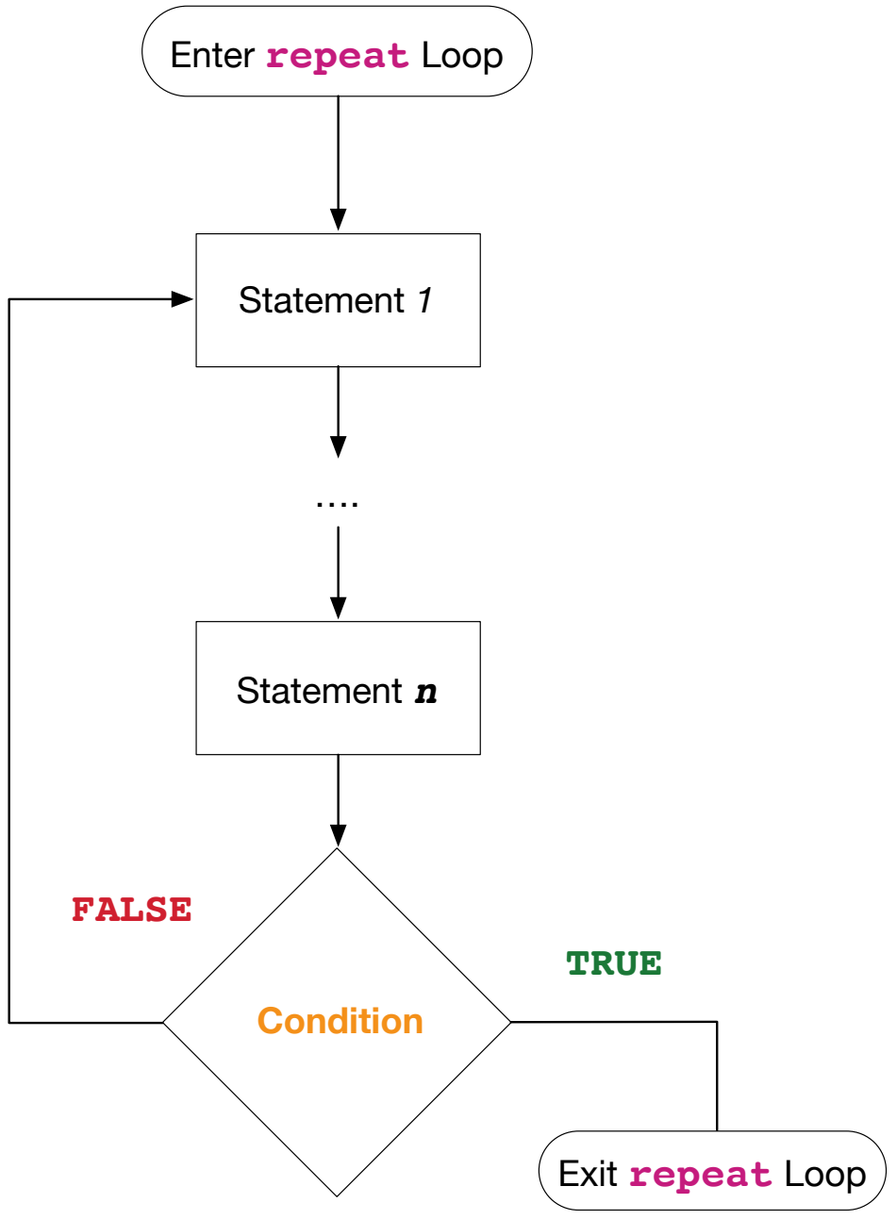
Nested Iteration

… non-symmetric matrix multiplication …

```
A = matrix(seq(1, 6), nrow = 2, ncol = 3)
B = matrix(seq(1, 6), nrow = 3, ncol = 2)
C = matrix(0, nrow = nrow(A), ncol = ncol(B))
if (ncol(A) != nrow(B)) {
  stop("matrices `A` and `B` dimensions are improper")
}

for (i in seq_len(nrow(A))) {
    for (j in seq_len(ncol(B))) {
        for (k in seq_len(ncol(A))) {
            C[i, j] = C[i, j] + A[i, k] * B[k, j]
        } # end third for loop
    } # end second for loop
} # end first for loop
```

repeat template

… do at least once …



Iteration Structure
Declaration of the iteration structure to use when perform repeat calculations

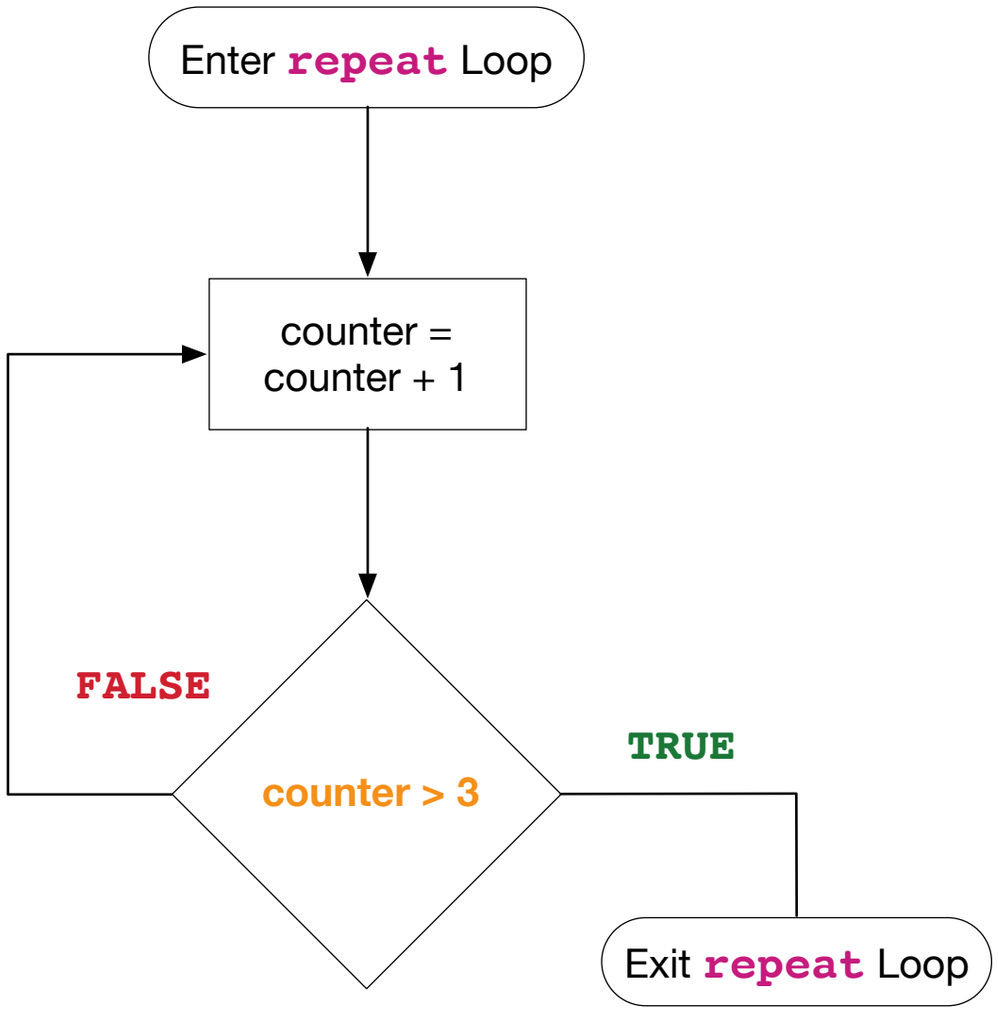
```
repeat {
    Statement 1
    .....
    Statement n
    if (<condition>) {
        break
    }
}
```

Body
Statements in between {} are run on each iteration of the loop

Logical Condition
Test to see whether the loop should go again (TRUE) or stop (FALSE)

Counting Up to **4**

… why not **3** ???



Initialization Statements
Statements used inside the *iteration* structure that must be defined prior to use

```
counter = 0
```

Iteration Structure
Declaration of the iteration structure to use when perform repeat calculations

```
repeat {
    counter = counter + 1
    if (counter > 3) {
        break
    }
}
```

Body
Statements in between {} are run on each iteration of the loop

Logical Condition
Test to see whether the loop should go again (TRUE) or stop (FALSE)

Example Force Response

… controlling user input …

```
repeat {
    input = readline("Who is the fairest of them all?")

    if (input == "JJB" || input == "Balamuta") {
        message("Correct! You're future looks brighter.")
        break # exits out of the repeat structure
    } else {
        message("I'm sorry Human, I'm afraid that's incorrect.")
    }
}
```

Your Turn

Add all of the numbers in a vector up until the first instance of an **NA**

```
x = c(9L, 88L, -2L, NA, 0L, NA)
_________ = _________________
_________ = _________________
repeat {
   if (   ___________________________ ) {
      _________
   }
   _________ = _________ ___ _________
}

_________
```

Recap

- **Iteration**
  - Forms of repeating the same instruction
  - Common structures: *for*, *while*, and *repeat*
- Special controls for inside an iteration structure
  - *break*: exit out of loop
  - *next*: go to the next value in loop.

Acknowledgements

Acknowledgements

- Bob Rudis' minimalist gestalt *C++* function diagram

- Hadley Wickham's *apply diagrams in Advanced R

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License