Control Structures and Flow

James Balamuta

Department of Informatics, Statistics
University of Illinois at Urbana-Champaign

June 22, 2017

Announcements

- Groups!
- **Reminder** HW1 Due June 25th @ 11:59 PM CDT
- HW2 assigned tomorrow and due July 2nd @ 11:59 PM CDT
- Lecture Feedback Survey for Week #2 Available Now at:
  https://goo.gl/forms/SQQlO3zZ9lc3j1o83

On the Agenda

1. Control Structures
   - Background
   - Sequential Control
   - Logical Operators
2. Selection Control
   - `if` and `if/else`
   - `if/elseif/else`
   - Special Selection Ops
   - `switch`
3. Iteration Control
   - `for` loops
   - `while` loops
   - `repeat` loops

On the Agenda

Control Structures and Flow of Control

- A **control structure** is a piece of code whose **analysis of a variable** results in a **choice being made as to the direction** the program should go.
- Meanwhile, the **Flow of Control** specifies the **direction** the program takes or **flows** when given conditions and parameters.

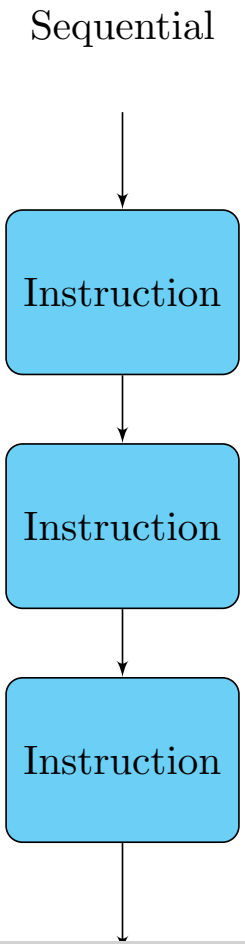
Types of Control Structures:

There are **three** different types of control structures.

- **Sequential:** Executes code statements one line after another
  - Exactly adhering to an algorithm like creating a cake from a recipe.
- **Selection:** Allows for decisions between 2 or more alternative paths.
  - Making a choice as to whether I want a Cold Brew or Iced Coffee from *Starbucks*.
- **Iteration:** Enables the looping or repeating of a section code multiple times.
  - Saying the same words over and over again.

Sequential Control

Sequential control is the default operating behavior of a computer program. The program will read each line one after another and execute it.



Logical Operators

To control the structure, we sometimes must make **decisions** that have different cases based on a *boolean*, `FALSE` (0) or `TRUE` (1), condition.

A few such *vectorized* logical operators are given as:

Operator	Explanation
`x == y`	`x` equal to `y`
`x != y`	`x` not equal to `y`
`x < y`	`x` less than `y`
`x > y`	`x` greater than `y`
`x <= y`	`x` less than or equal to `y`
`x >= y`	`x` greater than or equal to `y`
`x %in% y`	`x` is contained in `y`

Combining Logical Operators

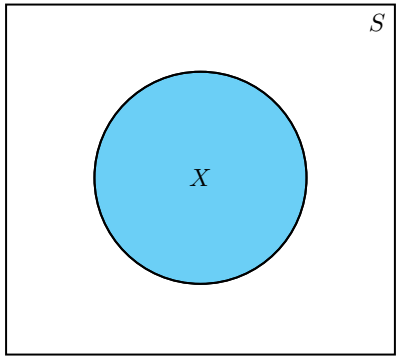
We can combine logical operators using:

Operator	Explanation
`!x`	not `x`
`xor(x,y)`	Exclusive or, only `x` or `y` is true
`x | y`	`x` or `y` (vectorized)
`x & y`	`x` and `y` (vectorized)

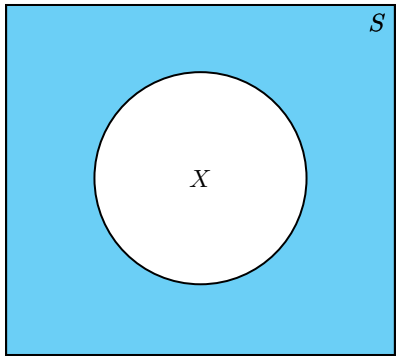
If we only want to compare *one* value (e.g. not vectorized), then we can use:

Operator	Explanation
`x || y`	`x` or `y` (not vectorized)
`x && y`	`x` and `y` (not vectorized)

Venn Diagrams of Logic: X and !X

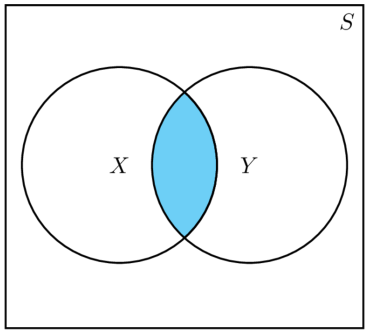


Area of X is the same as X

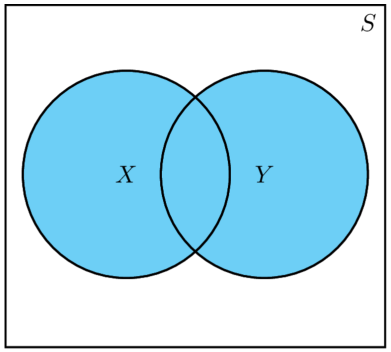


Area of X^c is the same as $!X$

Venn Diagrams of Logic: AND vs. OR

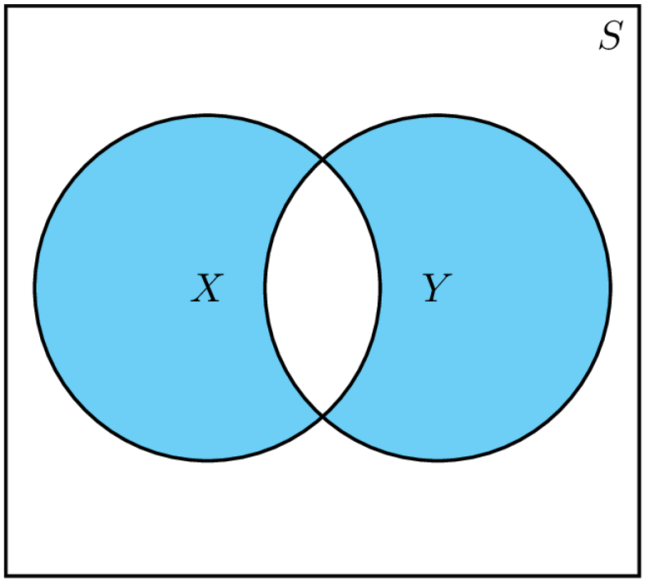


Area of $X \cap Y$ is the same as $X \&\& Y$



Area of $X \cup Y$ is the same as $X || Y$

Venn Diagrams of Logic: XOR



Area of $\overline{X \cap Y}$ is the same as $XOR(X, Y)$

Examples of Logical Operators

Explanation	Example	Result
x equal to y	`2 == 2`	`TRUE`
x not equal to y	`1 != 3`	`TRUE`
x less than y	`10 < 8`	`FALSE`
x greater than y	`10 > 8`	`TRUE`
x less than or equal to y	`c(6, 5) <= c(6, 1)`	`TRUE FALSE`
x greater than or equal to y	`c(6, 5) >= c(6, 1)`	`TRUE TRUE`
y contains x	`c(1,2) %in% c(3,1)`	`TRUE FALSE`
not x	`!(1 < 2)`	`FALSE`
Exclusive or, only x or y is true	`xor(c(T,T), c(T,F))`	`FALSE TRUE`
x or y (vectorized)	`c(F, F) | c(T, F)`	`TRUE FALSE`
x and y (vectorized)	`c(F, T) & c(F, F)`	`FALSE FALSE`
x and y (not vectorized)	`(1 < 10) && (15 > 10)`	`TRUE`
x or y (not vectorized)	`FALSE || TRUE`	`TRUE`

Practice with Logical Operators

Consider

```
x = seq(0, 2, by = 0.5)
y = 2*x - 1

# Scenario 1:
x < 1

# Scenario 2:
y == 1

# Scenario 3:
!(x < 1 | y == 1)
```

What is the end result?

Short Circuit Evaluation

- The && and || operators both have a feature called **short-circuit evaluation**.
- Consider the && (AND) expression (x && y) ...
  - If x is FALSE, then there is no reason to evaluate y so the evaluation stops.
  - Take for example a division check for zero:

```
x = 0; y = 4           # Define Variables
(x != 0 &&  y/x > 0) # Evaluates only x != 0
                       # so y/x never runs.
```

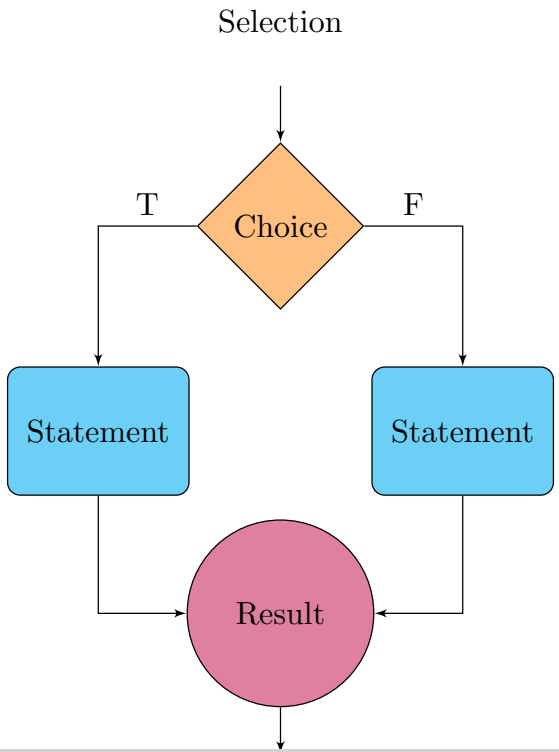
- Similarly, the || (OR) expression (x || y) ...
  - If x is TRUE, then whole expression is TRUE so there is no need to evaluate y.
  - Only in the case when x is FALSE does y get evaluated.

On the Agenda

1. Control Structures
   - Background
   - Sequential Control
   - Logical Operators
2. Selection Control
   - `if` and `if/else`
   - `if/elseif/else`
   - Special Selection Ops
   - `switch`
3. Iteration Control
   - `for` loops
   - `while` loops
   - `repeat` loops

Selection Control

- **Selection:** Allows for decisions between 2 or more alternative paths.
  - `if`
  - `if/else`
  - `if/elseif/else`
  - `switch`



Selection Control - if and if/else

The if statement provides the ability to execute code only when the condition is TRUE like so:

```
if (condition) {# True Case
  # statement
}
```

Traditionally, one would normally use if/else statement to address both TRUE and FALSE cases:

```
if (condition) {# True Case
  # statement
} else {          # False Case
  # statement
}
```

Note: The condition clause of an if statement is *not* vectorized.

Selection Control - if and if/else Examples

```
x = 2
if (x > 0) {  # Detect if x is a positive number
  cat(x)      # Print `x` to console
}
```

```
## 2
```

```
if (x != 42) {   # True Case
  cat("This is not the meaning of life")
  x = 42
} else {         # False Case
  cat("This is the meaning of life!")
}
```

```
## This is not the meaning of life
```

Selection Control - if and if/else: Concept Check

Identify the problematic areas of the following if statements in *R*:

Example 1:

```
x = 9; y = 20
if (x < 10) && (y < 35)
  cat("Bingo was his name!\n")
```

Example 2:

```
x = 43
if (x == 42 || 43 || 44)
  cat("That's close to my age!\n")
```

Selection Control - if and if/else: Concept Check (Pt 2)

Example 3:

```
x = seq(0, 1, by = 0.1)
y = seq(.1, 1.1, by = 0.1)
if (x < y)
  TRUE
```

Selection Control - Vectorized if/else

In the case of the last `if` statement, we have the ability to use *R*'s vectorized `ifelse()`

```
ifelse(condition, TRUE-CONDITION, FALSE-CONDITION)
```

Example:

```
x = seq(0, 1, by = 0.2)
y = seq(-1, 3, by = 0.7)
ifelse(x < y, TRUE, FALSE)
```

```
## [1] FALSE FALSE FALSE  TRUE  TRUE  TRUE
```

Note: Watch out for vector recycling!

An alternative approach: Simplify the logical vector.

R has two functions that allow for the simplification of *logical* traits:

- `any()`: Only *one* value must be `TRUE` for the `if` to be `TRUE`.
- `all()`: All values must be `TRUE` for the `if` to be `TRUE`.

Simplifying the logical vector

Examples:

```
x = 1:5
any(x < 2)
```

```
## [1] TRUE
```

```
all(x > 3)
```

```
## [1] FALSE
```

```
if(all(x > 0)) cat("Greater than 0") else cat("Less than 0")
```

```
## Greater than 0
```

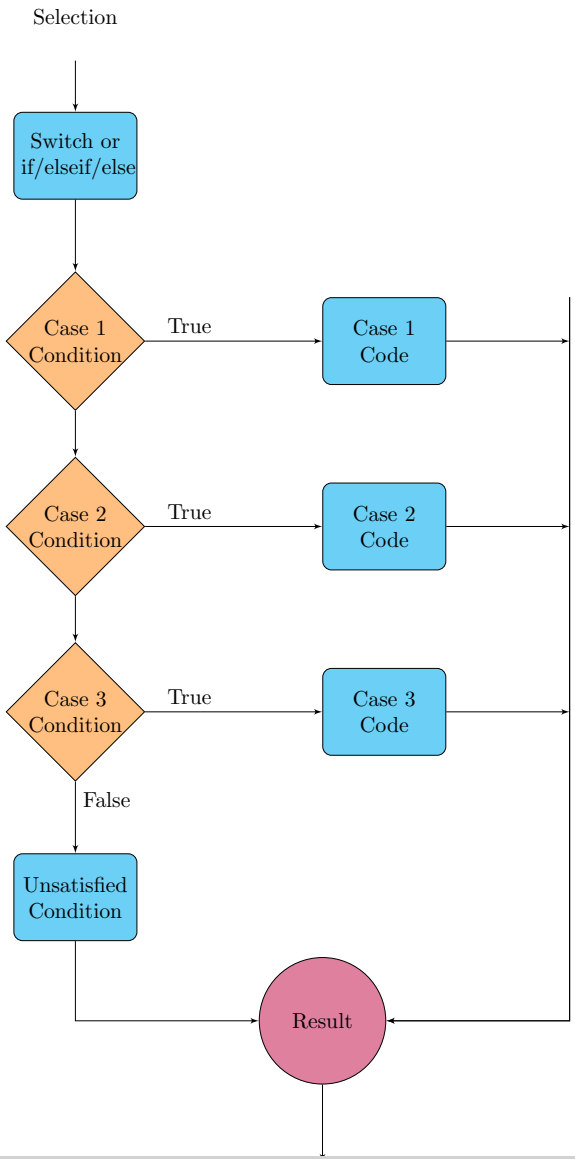
Selection Control - if/elseif/else

Sometimes, we may wish to group together certain conditions

```
if(condition) {
    # statements
} else if (condition2) {
    # statements
} else {
    # statements
}
```

In this case, the first statement is checked, if it does not contain the correct value the next statement is check and so until either one of the `conditions` is true or it arrives at the `else` condition.

Selection Control - if/elseif/else



Selection Control - if/elseif/else

Example:

```
x = 3
if (x == 42) {
  cat('Equal\n')
} else if (x > 42) {
  cat('Greater Than\n')
} else {
  cat('Less Than\n')
}
```

```
## Less Than
```

Selection Control - if/elseif/else

The if/elseif/else could be written using only if/else

```
x = 3
if (x == 42) {
  cat('Equal\n')
} else {
  if (x > 42) {
    cat('Greater Than\n')
  } else {
    cat('Less Than\n')
  }
}
```

```
## Less Than
```

Discussion: Why might structuring the if this way be not so ideal?

Selection Control - Vectorized `if/elseif/else`

Following in the previous examples footsteps, we can write a vectorized version to process multiple observations via:

```
ifelse(x == 42, 'Equal',
       ifelse(x > 42, 'Greater Than', 'Less Than'))
```

```
## [1] "Less Than"
```

Selection - Ternary Operator

Previously, we saw a *vectorized* version of the ternary operator given by `ifelse()`.

What makes a selection statement a **ternary operator** is it is meant to be an **inline** `if/else` statement. e.g.

```
if(condition) TRUE  else  FALSE
```

Consider:

```
x = 42
a = if(x == 42) TRUE else FALSE # Verbose
b = (x == 42)                   # Concise
all.equal(a, b)
```

Selection Control - switch

Within a switch, the condition is evaluated as the first parameter and then compared against the values in the case labels.

```
switch(type,
       case_1 = statement_1,
       case_2 = statement_2,
       statement_3 # Default / Else
)
```

Switch are often represented as if/elseif/else like:

```
if (type == case_1) {
  # statement 1
} else if (type == case_2) {
  # statement 2
} else {
  # statement 3
}
```

Selection Control - switch Example

There are multiple ways to switch in R. Principally, switches either respond to **index** or **named condition**.

Index:

```
switch(1,
       "First",
       "Second")
```

```
## [1] "First"
```

Named condition:

```
switch("toad",
       prince = "First",
       toad = "Second",
       "Third")
```

```
## [1] "Second"
```

Note: Faster to use a `switch` than an `if/elseif/else` in *R*!

Selection Control - switch: Concept Check

Consider the following switch:

```
switch(2,
       prince = "First",
       toad = "Second",
       "Third")
```

- What will be the output?

Selection Control - switch: Concept Check

Let's make a modification to the previous switch to use an index, e.g.

```
switch(4,                    # Switched to Numeric
       prince = "First",
       toad = "Second",
       "Third")
```

What will be the output?

Selection Control - switch: Concept Check

- What happens when we change to use a named condition?

```
switch("Fourth",                    # Switched to Named
       prince = "First",
       toad = "Second",
       "Third")
```

- How does this jive with the previous switch example?

On the Agenda

1. Control Structures
   - Background
   - Sequential Control
   - Logical Operators
2. Selection Control
   - `if` and `if/else`
   - `if/elseif/else`
   - Special Selection Ops
   - `switch`
3. Iteration Control
   - `for` loops
   - `while` loops
   - `repeat` loops

Iteration Control

- **Iteration:** Enables the looping or repeating of a section code multiple times.
  - `for`
  - `while`
  - `do/while`

Iteration Control

> *"I choose a lazy person to do a hard job. Because a lazy person will find an easy way to do it."*
> — *Bill Gates*

Iteration is the advent of the ability to repeat statements like:

```
cat("Hello World!\n")
cat("Hello World!\n")
cat("Hello World!\n")
```

Without having to type it!

Iteration Control - for

The most common loop in *R* is that of the `for` loop. The `for` loop is structured as:

```
for(variable in vector) {
  # statements
}
```

Typically, the `vector` component is called the *loop index* and is an *integer*. However, *R* also allows it to be vector names or actual data.

Iteration Control - for: Summation Example

Consider the summation example: $\sum_{i=1}^{n} x_i$

```
n = 5                          # Amount to iterate over
set.seed(111)                  # Set seed for reproducibility
x = runif(n)                   # Generate n data points
summed = 0                     # Output Variable
for (i in 1:n) {               # Iteration Sequence
  summed = summed + x[i]       # Statement
}
```

Iteration Control - for

The for loop provides the ability to **automatically** increment the counting variable or iterate through a set of named expressions.

```
x = c("coffee","doppio espresso",
      "iced coffee", "cold brew")

# Numeric Index
for (i in 1:length(x)) {
  cat("i is: ", i, ", x[i] is:", x[i],"\n", sep= "")
}

# Named Index
for (i in x) {
  cat("i is: ", i, "\n", sep= "")
}
```

Iteration Control - Index Examples

Consider the loop of:

```
a = numeric(0)
for(i in 1:length(a)){
  cat("Hello!\n")
}
```

1. What *should* the output be?
2. What *is* the output?

Iteration Control - Index Protection

Instead of using `1:length(a)` to iterate a loop counter, define `n = length(a)` and try to use: `seq_len(n)` or `seq_along(a)`

```
a = numeric()
n = length(a)

# Case 1
for(i in 1:n) { cat("Hello Unprotected Loop") }

# Case 2
for(i in seq_len(n)) { cat("Hello Protected Loop v1!\n") }

# Case 3
for(i in seq_along(a)) { cat("Hello Protected Loop v2!\n") }
```

Iteration Control - Index Protection

- In essence, the error in **Case 1** is due to `length(a)` being `0`.
- This leads the sequence `1:length(a)` to be `1:0`, which creates the vector `c(1,0)`.
- In this case, the `for` loop is then able to iterate.
  - If we had decided to *subset* a value, then an error would be thrown!

Iteration Control - for skipping

Sometimes, we might have a case that we want to *skip* in the loop. To do so, we use the `next` statement to move the loop forward.

```
for (i in 1:4) {
  if(i %% 2 == 1) {
    next       # skip odd numbers
  }
  cat(i, "\n")
}
```

```
## 2
## 4
```

Iteration Control - while

Meanwhile, a while loop requires three items:

1. condition,
2. statements,
3. change logic

```
while (condition) {
  # statements

  # change logic
}
```

Iteration Control - while

Take for example the following loop:

```
i = 1
while (i < 3) {
  cat(i, "\n")
  i = i + 1     # Do not forget to increment!
}
```

This is a perfect candidate for looping as long as we remember to **increment** the looping variable.

Checking in with a while

What is the output of the following piece of code?

```
y = 0
x = 10
i = 0

while(i < x) {
  if(i %% 2 == 0) {
    i = i + 1
  } else if (i %%3 == 0) {
    i = i + 2
  } else {
    y = y + i
  }
}

y == 18
```

Iteration Control - break out from a loop

We may obtain the desired result before the end of the loop and, thus, no longer need to continue the iteration.

To escape from a loop, we need to use `break`.[1]

```
i = 1
while (i < 5) {
  if(i == 3) { # Stops the loop at the 3rd iteration.
    break
  }
  cat(i, "\n")
  i = i + 1     # Do not forget to increment!
}
```

[1] **Note:** Break also works in a `for` loop.

Iteration Control - repeat or a do-while

Previously, for both `for` and `while` we had to have the initial condition met before the loop would run. However, sometimes we may want to let code run until a specific threshold is met (e.g. for a simulation).

```
x = 1
y = 10
ntrials = 0
repeat {

  if(runif(1) > 0.5) {
    x = x + 1
  }

  ntrials = ntrials + 1

  if(x < y) {
    break
  }
}
```

What is the output of `ntrials < x`?

Summary of Control Structures

- Discussed logical conditions
- Explored the different levels of selection controls `if`, `if/else`, `if/elseif/else`, `switch`
- Talked about the different types of R loops: `for`, `while`, and `repeat`
- Emphasized proper loop indexing techniques.