Introduction to Programming
Logical Expressions & Conditionals

Hye-Chung Kum
Population Informatics Research Group
http://research.tamhsc.edu/pinformatics/
http://pinformatics.web.unc.edu/

License:
Data Science in the Health Domain by Hye-Chung Kum is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License

Course URL:
http://pinformatics.tamhsc.edu/phpm672
What we are going to learn

- Operators
  - Logical (~ / !), (& / and), (| / or)
  - Relational (<, <=, ==, >, >)

- Learn Conditional programming
  - if then else end

- Common Pitfalls
# Relational Operators
Tests relationship between two objects

<table>
<thead>
<tr>
<th>Name</th>
<th>Operators</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Equivalence</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Equality</td>
<td>= (SAS)</td>
<td>5 == 5, x == y</td>
</tr>
<tr>
<td></td>
<td>== (STATA)</td>
<td></td>
</tr>
<tr>
<td>Inequality</td>
<td>~= (SAS)</td>
<td>5 ~= 5, z == (x^2 + y^2)</td>
</tr>
<tr>
<td></td>
<td>!= (STATA)</td>
<td></td>
</tr>
<tr>
<td><strong>Binary Operators</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Less Than</td>
<td>&lt;</td>
<td>5 &lt; 3</td>
</tr>
<tr>
<td>Less Than or Equal</td>
<td>&lt;=</td>
<td>4 &lt;= 4,</td>
</tr>
<tr>
<td>Greater Than or Equal</td>
<td>&gt;=</td>
<td>7 &gt;= 10</td>
</tr>
<tr>
<td>Greater Than</td>
<td>&gt;</td>
<td>10 &gt; 7</td>
</tr>
</tbody>
</table>
Logical Operators

Boolean operators

<table>
<thead>
<tr>
<th>Name</th>
<th>Operators</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unary Operators</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logical Negation (NOT)</td>
<td>~ (SAS) / ! (STATA)</td>
<td>~ (3 == 5) = 1 (true)</td>
</tr>
<tr>
<td><strong>Binary Operators</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logical And (AND)</td>
<td>&amp; / and (SAS)</td>
<td>T &amp; T = 1 (true)</td>
</tr>
<tr>
<td>Logical Or (OR)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Performs binary logic on two logical data type operands to return a logical result.
## Boolean Logic

### Truth Tables (1=T; 0=F)

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>NOT</th>
<th>AND</th>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Logical Expressions

- Simple or complex expression whose final result is a single true/false logical result

**Examples:** Given \( x=3 \), \( y=4 \), \( z=5 \)
  - \( x == 3 \)
  - \((x+y) < z\)
  - Logical operators allow us to build up compound tests, piece by piece
## Operator Precedence (Full)

<table>
<thead>
<tr>
<th>Level</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(highest) Parentheses ( ) inner to outer</td>
</tr>
<tr>
<td>2</td>
<td>Transpose ′, Power ^,</td>
</tr>
<tr>
<td>3</td>
<td>Unary plus +, Unary Minus -, logical negation ~</td>
</tr>
<tr>
<td>4</td>
<td>Multiplication *, Division /</td>
</tr>
<tr>
<td>5</td>
<td>Addition +, Subtraction −</td>
</tr>
<tr>
<td>6</td>
<td>Comparisons &lt; , &lt;=, &gt;, &gt;=, ==</td>
</tr>
<tr>
<td>7</td>
<td>Logical ‘And’ &amp;</td>
</tr>
<tr>
<td>8(lowest)</td>
<td>Logical ‘Or’</td>
</tr>
</tbody>
</table>

* Left to right rule applies

- \( x \ & \ y \ | \ z = ? \) (put parenthesis)
### Boolean Logic

#### Truth Tables: \( x \ & \ y \ | \ z \)

| x | y | z | x & y | (x&y)|z| (y|z) | x&(y|z) |
|---|---|---|-------|--------|---|-------|--------|
| 0 | 0 | 0 | 0     | 0      | 0 | 0     | 0      |
| 0 | 0 | 1 | 0     | 1      | 1 | 1     | 0      |
| 0 | 1 | 0 | 0     | 0      | 1 | 0     | 0      |
| 0 | 1 | 1 | 0     | 1      | 1 | 0     | 0      |
| 1 | 0 | 0 | 0     | 0      | 0 | 0     | 0      |
| 1 | 0 | 1 | 0     | 1      | 1 | 1     | 1      |
| 1 | 1 | 0 | 1     | 1      | 1 | 1     | 1      |
| 1 | 1 | 1 | 1     | 1      | 1 | 1     | 1      |
Logical Data Types

- **Data Range**
  - Conceptually: Takes on only two Values
    - *true or false* (1 or 0)
  - Actually:
    - *false* ↔ *zero* (0)
    - *true* ↔ any non-zero value (1 or greater)
    - This difference can cause subtle bugs if you are not careful.

- **Storage**
  - Conceptually: Uses a single binary bit
  - Physically/Actually: Takes a single byte
Other Logical Objects

- Functions which return logical data types as their output
- Test functions (is* functions)
  - Examples: isfloat(), isvarname(), iskeyword()
- String Comparison functions:
  - strcmp(), strcmpi(), strncmp(), strncmpi()
Motivation

- Step by Step Programming
  - All we have learned to do up to now...
  - Execute statements in order they occur
  - Single path through program script

- Conditional Programming
  - What if we only want to run the code only if some test is satisfied? (print if cond)
  - What if need to make a choice between 2 or more options?
  - How do we make the choice?
Example

**SAS**

* Initialize to default hourly rate;  
  * If MS, assign higher rate;

```sas
rate=10;
if edu>3 then rate=12;
proc print data=fn(obs=10);
where gender='F';
```

```
If-end Statement
Single conditional path

• Syntax:

```plaintext
if <test> then [do;]
       commands; * 1 or more;
[end;]
```

• **Tip:** For the `<test>`, use logical expressions that evaluate to a single `true/false` value.
Simple Example

* One way;
rate=10;
if (edu > 3) then do;
    rate=12;
end;

* Another way;
rate=10;
if (edu > 3) then rate=12;
If-else-end statement
Two alternatives, if <true> else <false> end

• Syntax:

```plaintext
if <test> then [do;]
  commands1; * True;
end; else do;
  commands2; * False;
end;
```
**Simple Example**

* One way;

```plaintext
if (edu > 3) then do;
    rate=12;
end; else do;
    rate=10;
end;
```

* Another way;

```plaintext
if (edu > 3) then rate=12;
else rate=10;
```
If-elseif-else-end Conditional Execution

Multiple chained tests

```plaintext
if <Test1> then do;
  commands1; * T1 true;
end; else if <Test2> then do;
  commands2; * T2 true;
end; else if <Test3> then do;
  commands3; * T3 true;
end; else do;
  commands4; * all false;
end;
```
Example:

```plaintext
if (edu > 5) then do;
    rate=16;
end;
else if (edu > 4) then do;
    rate=14;
end;
else if (edu > 3) then do;
    rate=12;
end;
else do;
    rate=10;
end;
```
Conditional Execution
Nested conditions

```plaintext
if <Test1> then do;
  if <Test2> then do;
    commands1; * T1,T2 both true;
  end; else do;
    commands2; * T1=1, T2=0;
  end;
end; else do;
  if <Test3> then do;
    commands3; * T1=0, T3=1;
  end; else do;
    commands4; * T1,T3 both false;
  end;
end;
```
if expression

statements

end
if expression
    statements 1
else
    statements 2
end
if \( \text{exp1} \)
  \text{statements 1}
else if \( \text{exp2} \)
  \text{statements 2}
else
  \text{statements 3}
end
while (expression) do;
  statements;
end;

do (expression) until;
  statements;
end;
Common Pitfalls

- Using `=` instead of `==` and vice versa.
  - SAS: same, STATA: different
  - `if x = 5` ... % Error, use `if x == 5`

- Confusing `&` (and) and `|` (or)

- Inserting an extra space in a 2 character relational operator
  - `if x <= y` % Error, note extra space
  - `if x <= y` % Correct
Common Pitfalls, cont.

- Using multiple comparisons properly
  - $10 \leq x \leq 100$  \text{% Error (OK in SAS)}
  - $(10 \leq x) \& (x \leq 100)$  \text{% Correct}

- Forgetting the quotes when working with characters or strings
  - 
    - \text{if letter} ==y  \text{% Error (y is the name of var)}
    - \text{if letter} ==”y”  \text{% Correct (y is value of var)}

- Comparing characters / strings (be careful)
  - 'c' < 'Z'  \text{% OK, compatible sizes}
  - 'cat' < 'catch'  \text{% Error, size problem}
  - \text{strcmp}('cat', 'catch')  \text{% Use strcmp}
Common Pitfalls, cont.
using if ... end instead of if ... else .. end

if (error)
    disp(errMsg);
else
    ...  %Continue
end

- Despite detecting an error, we continue on to execute the rest of the script or function

- We only execute the rest of the script or function, if we are error free.
Logical Expressions & Conditional Programming
Reminder

• Practice using conditional logic
  ◦ Learn logical operators  ~, &, |
  ◦ Learn relational operators  <, <=, ==, >, >=
  ◦ Logical expressions
  ◦ If statement

• Practice writing conditional code

• Do the online modules
Learn to fish

- Reading: READ sections in the recommended book & modules I give you before class
- Give you good problems (lab & assignment) to learn to fish on your own
  - Lab: Read my/TA code
  - Assignment: Now write your code
- Available when you get stuck
- Top (problem) down(data) vs bottom up
  - Need to iterate
Before we start

- I will do more coding in class so you can see how coding is done
  - Remember this is just ONE way of doing it. I have very old habits from when computers were very different. So pick and choose what you think works for you
- LAB: I will share code I write, so you learn to read code
- Assignment: now try to write code to do similar things with your own data
- Computing environment is important
  - Does everyone have a stable environment?
  - Any question?