Chapter 3

Selection Statements
What we have done so far:

- Scripts/Functions have executed all commands in order, not matter what

What we often need:

- A piece of code that executes a series of commands, if and only if some condition is met

- MATLAB provides several built-in statements that allow for conditional behavior
  - if/elseif/else
  - switch
  - menu
To make selections, we must be able to determine if a condition is met
  - Boolean expression
  - Logical expression
  - Relational expression

A new class of variable is required
  - Logical
  - Also often called “Boolean Logic”
# Truth Table for Logical Operators

- Logical operators...follow basic logic!

<table>
<thead>
<tr>
<th>Table 3.1</th>
<th>Truth Table for Logical Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 3.2</th>
<th>Operator Precedence Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operators</td>
<td>Precedence</td>
</tr>
<tr>
<td>parentheses ()</td>
<td>highest</td>
</tr>
<tr>
<td>transpose and power ‘, ^</td>
<td></td>
</tr>
<tr>
<td>unary negation (-), not (~)</td>
<td></td>
</tr>
<tr>
<td>multiplication, division *, /, \</td>
<td></td>
</tr>
<tr>
<td>addition, subtraction +, -</td>
<td></td>
</tr>
<tr>
<td>colon operator:</td>
<td></td>
</tr>
<tr>
<td>relational &lt;, &lt;=, &gt;, &gt;=, ==, !=</td>
<td></td>
</tr>
<tr>
<td>and &amp;&amp;</td>
<td></td>
</tr>
<tr>
<td>or</td>
<td></td>
</tr>
<tr>
<td>assignment =</td>
<td>lowest</td>
</tr>
</tbody>
</table>
Relational/Boolean Expressions

• Logical variables
  • Only two possible values
  • 1 (true)
  • 0 (false)

• Lets look at some sample Boolean expressions
Relational/Boolean Expressions

- Relational operators can be used on characters
  - MATLAB uses the ASCII numeric value
  - Not typical usage
  - You should know this is possible for error de-bugging
- Even less common to use on strings

```matlab
>> 'a' < 'd'
```

```matlab
>> int8('a')
```

```matlab
>> int8('d')
```

```matlab
>> 'a' < 'A'
```

```matlab
>> int8('A')
```
In MATLAB, we can use logical variables in mathematical operations.

- Typically, this is not a good idea.
- You should know this is possible for error debugging.
Relational/Boolean Expressions

• Don’t do stuff like this in your code!
• I only show here, so we can understand how MATLAB treats these strange Boolean expressions

\[(\text{Boolean } - \text{math}) > (\text{Boolean } + \text{math})\]
The If Statement

• MATLAB provides an “if” statement
  • Nearly all programming languages have something similar
  • Can be entered in the command window
  • More commonly used in scripts/functions

The “if” keyword (“if” is a reserved word in MATLAB)

A logical or relational condition

Warning: Only works for scalars

Actions to be performed if condition(s) is(are) TRUE

Note: The actions should be indented (easier to read)

The “end” keyword that ends the “if” statement

```
if condition
  action(s)
end
```
• Be sure to read Attaway about how non-zero values behave in if statements.
  • Any non-zero number in a logical statement -----> true
Multiple If Statements

- If two conditions can both be true, or false, or one of each
- May need multiple if statements

```matlab
% This is an example of a case where multiple if statements are needed
%multiply.m
x=24;
if rem(x,2) == 0
    fprintf('%d is divisible by 2\n',x);
end
if rem(x,3) == 0
    fprintf('%d is divisible by 3\n',x);
end
if rem(x,4) == 0
    fprintf('%d is divisible by 4\n',x);
end
if rem(x,5) == 0
    fprintf('%d is divisible by 5\n',x);
end
if (rem(x,2) == 0) && (rem(x,3) == 0) && (rem(x,4) == 0) && (rem(x,5) == 0)
    fprintf('%d not divisible by 2, 3, 4, or 5\n',x);
end
```

We will find a better way to do this later...
The If-Else Statement

- What if more than one condition needs to be tested?
  - Use a nested if-else statement
  - Can execute different actions depending on if condition is met
  - More efficient than separate if statements

```
if condition
  action(s)
else
  action(s)
end
```

- **Actions only executed if condition is TRUE**
- **Actions only executed if condition is FALSE**
Poor Usage of If Statements

- Don’t do this!
  - It may work, but this is poor coding

- Each if statement must always be checked
  - Inefficient

- Use nested if-else statements
Proper Use of Nested If-Else Statement

• Same thing, but using nested if-else statements
  • More efficient
  • Better style
  • Easier to read

```matlab
% an example of proper usage of a nested if-else statement
% betterIf.m

x=14;
if x == 15
    fprintf('%d is 15\n',x);
else
    fprintf('%d is not 15\n',x);
end
```

```matlab
>> betterIf
14 is not 15
```

```matlab
>> fx
```

```matlab
```
The If-Elseif-Else Statement

- What if multiple conditions need to be tested
  - Each one results in different actions
- Use a nested if-elseif-else statement
  - MUCH more efficient than separate if statements
  - Can have as many elseif statements as needed

```java
if condition1
  action(s)
elseif condition2
  action(s)
else
  action(s)
end
```

- Only executed if condition1 is true
- Only executed if condition1 is FALSE and condition2 is TRUE
- Only executed if condition1 and condition2 are BOTH FALSE
More than one of these conditions tested can never be true at the same time.

- “mutually exclusive”

There is a more elegant and efficient way to code this.

Nested if, elseif, else statements

Also, some of these conditions do not need to be tested.
Proper Use of Nested If-Elseif-Else

- If conditions are mutually exclusive
  - Use nested if, elseif, if

- Nested if statements save space and CPU time

- Also, much easier to read

```matlab
% This is an example of proper usage of nested if statements

x = 7;

if x > 10
    fprintf('x is greater than 10\n');
elseif x >= 5 && x <= 10
    fprintf('x is between 5 and 10\n');
elseif x >= 0 && x < 5
    fprintf('x is between 0 and 5\n');
elseif x < 0
    fprintf('x is negative\n');
end
```

MATLAB demo: betterIf2.m guessNum.m
Turn scripts into functions
Even Better Use of Nested If-Elseif-Else

- The last elseif does not need to be tested
- Just use else
- Some of the conditions do not need to be tested
  - Redundant testing slows down code
  - This code is efficient

```matlab
% This is an example of proper usage of nested if statements
% x = 7;

if x > 10
    fprintf('x is greater than 10\n');
elseif x >= 5
    fprintf('x is between 5 and 10\n');
elseif x >= 0
    fprintf('x is between 0 and 5\n');
else
    fprintf('x is negative\n');
end
```

```
>> betterIf3
    x is between 5 and 10
```
The Switch Statement

- What if many conditions need to be tested?
  - Each one results in different actions
- A nested if-elseif-else statement can get tedious to write
- Use a switch statement
  - Tests each condition separately
  - Conditions must be mutually exclusive

<table>
<thead>
<tr>
<th>Reserved word, switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>The variable to be tested</td>
</tr>
<tr>
<td>Condition to test “variable”</td>
</tr>
<tr>
<td>If true, action(s) are performed</td>
</tr>
<tr>
<td>If first condition fails, test this condition. If true, do stuff.</td>
</tr>
<tr>
<td>If second condition fails, test this condition. If true, do stuff.</td>
</tr>
<tr>
<td>If all conditions fail, do stuff.</td>
</tr>
</tbody>
</table>
Tedious Nested If Statement

- A function that returns the day of the week (a string), given a numeric input
- There is an easier way to write this same thing
  - Use a switch statement

```
function day = dayOfWeekIf(num)
  % returns a string with the day of the week
  % only works for an integer input 1-7
  if num==1
    day='Sunday';
  elseif num==2
    day='Monday';
  elseif num==3
    day='Tuesday';
  elseif num==4
    day='Wednesday';
  elseif num==5
    day='Thursday';
  elseif num==6
    day='Friday';
  elseif num==7
    day='Saturday';
  else
    day='Error';
  end
end
```

```
>> dayName = dayOfWeekIf(3)
dayName =
Tuesday
>> dayName = dayOfWeekIf(6)
dayName =
Friday
>> dayName = dayOfWeekIf(62.4)
dayName =
Error
```
• Same as before, but using “switch”

• The switch statement is easier when a lot of choices are possible

```matlab
function day = dayOfWeekSwitch(num)
    % returns a string with the day of the week
    % only works for an integer input 1-7
    switch num
    case 1
        day='Sunday';
    case 2
        day='Monday';
    case 3
        day='Tuesday';
    case 4
        day='Wednesday';
    case 5
        day='Thursday';
    case 6
        day='Friday';
    case 7
        day='Saturday';
    otherwise
        day='Error';
    end

% Example usage:
>> dayName = dayOfWeekSwitch(3)
dayName =
    'Tuesday'

>> dayName = dayOfWeekSwitch(6)
dayName =
    'Friday'

>> dayName = dayOfWeekSwitch(62.4)
dayName =
    'Error'
```
The Menu Function

- If you are writing code that may be used by novice users, often having graphical buttons is useful
  - Not good for expert users
  - Cannot easily automate mouse clicks
  - Clicking buttons is very slow

- It is good to understand that buttons in software are executing source code

- MATLAB provides the “menu” function
  - Makes simple graphical buttons
  - Pops up in a new window
  - Pairs up well with switch or nested if statements
The menu function is straightforward and easy to use.

```matlab
choice = menu('instructions','option1','option2','option3')
```

- **Where the returned value (i.e. the choice) is stored**
- **Text/Instructions to put in the menu box above the buttons**
- **Text to put on top button**
  - If clicked, returns 1
- **Text to put on second from top button**
  - If clicked, returns 2
- **Text to put on third from top button**
  - If clicked, returns 3

*MATLAB demo: menu function with no script*
% A simple example of the menu function
% User gets three buttons to choose from
% How can otherwise get selected?

```matlab
% first script using the menu function
% firstMenu.m
%
pick = menu('What should I do?','Print a message','Plot a graph','Print my lucky number');
switch pick
    case 1
        fprintf('You rock at menus!
');
    case 2
        x=0:0.1:10;
        y=cos(x)+rand;
        plot(x,y,'r-o','MarkerFaceColor','r','MarkerEdgeColor','k','MarkerSize',3);
        fprintf('Here is your graph
');
    case 3
        num=round(rand*100);
        fprintf('Your lucky number is: %d
',num);
    otherwise
        fprintf('You didn''t select anything!
');
        fprintf('That makes me sad
');
        fprintf(':(
');
end
```
• Menu buttons are easy to make!
• Only good if you are writing code for novice users
• How did I get the last message to print?

```plaintext
% A first script using the menu function
firstMenu.m

pick = menu('What should I do?', 'Print a message', 'Plot a graph', 'Print my lucky number');
switch pick
    case 1
        fprintf('You rock at menus!\n');
    case 2
        x=0:0.1:10;
        y=cos(x)+rand;
        plot(x,y,'r-o','MarkerFaceColor','r','MarkerEdgeColor','k','MarkerSize',3);
        fprintf('Here is your graph\n');
    case 3
        num=round(rand*100);
        fprintf('Your lucky number is: %d\n',num);
    otherwise
        fprintf('You didn’t select anything!\n');
        fprintf('That makes me sad\n');
        fprintf(': (\n');
end
```

```
>> firstMenu
You rock at menus!
>> firstMenu
Here is your graph
>> firstMenu
Your lucky number is: 10
>> firstMenu
You didn’t select anything!
That makes me sad
```
A simple example of the menu function
User gets three buttons to choose from
- How can else get selected?

```matlab
% A first script using the menu function
% firstMenu.m

pick = menu('What should I do?','Print a message','Plot a graph','Print my lucky number');
if pick==1
    fprintf('You rock at menus!\n');
elseif pick==2
    x=0:0.1:10;
    y=cos(x)+rand;
    plot(x,y,'r-o','MarkerFaceColor','r','MarkerEdgeColor','k','MarkerSize',3);
    fprintf('Here is your graph\n');
elseif pick==3
    num=round(rand*100);
    fprintf('Your lucky number is: %d\n',num);
else
    fprintf('You didn''t select anything!\n');
    fprintf('That makes me sad\n');
    fprintf(':('\n');
end
```
Menu Function Example (Part II)

- “menu” can also be used with nested if statements

MATLAB demo:
firstMenu.m; firstMenu2.m
Add buttons to guessNum.m
• Practice Boolean logic
  • Recall order of operations in MATLAB
  • Makes sense, but only if you practice

**Selection statements are powerful**

• Make sure to use the most logical and efficient method
  • Use nested if-elseif-else statements if choices are mutually exclusive
  • If lots of integer choices, consider a switch statement
  • Indent actions in selection statements! (be neat)

• “menu” provides a nice graphical window with buttons for the user to click
  • Difficult to automate
  • Only use if code is for small data, or novice users
Chapter 4: Loop Statements

• How to program data loops
• How to do selection statements on large data sets
• Can do powerful stuff
• This is when stuff gets fun!