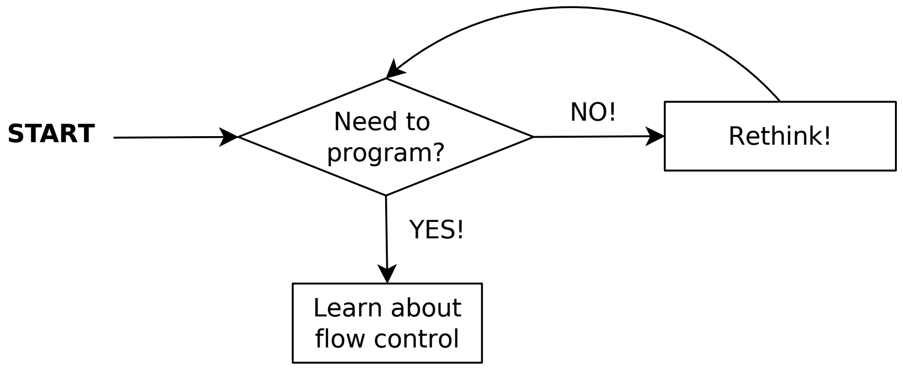
Fundamental Programming Principles: Flow Control



Beyond the Mouse

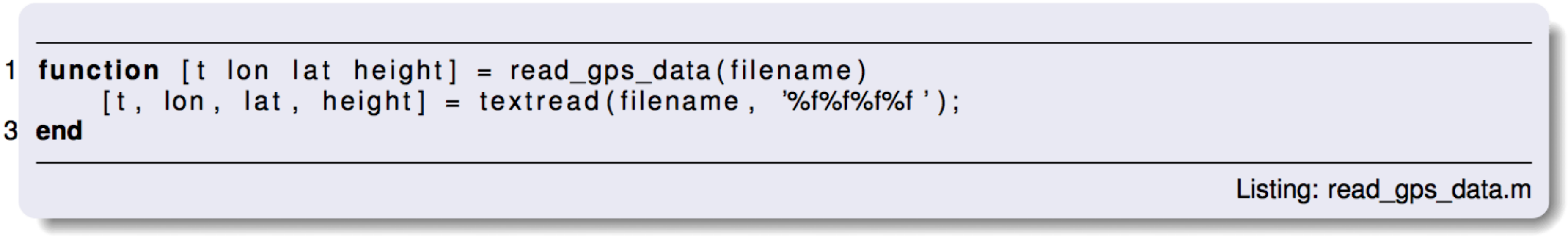
GEOS 436/636

Jeff Freymueller, Sep 19, 2017

YOU'LL NEVER FIND A PROGRAMMING LANGUAGE THAT FREES YOU FROM THE BURDEN OF CLARIFYING YOUR IDEAS.
BUT I KNOW WHAT I MEAN!

"The Uncomfortable Truths Well", http://xkcd.com/568 (April 13, 2009)

Intro – Do You Recognize This?



```
function [t lon lat height] = read_gps_data(filename)
    [t, lon, lat, height] = textread(filename, '%f%f%f%f');
end
```

Listing: read_gps_data.m

Intro – Do You Recognize This?

```matlab
function [t lon lat height] = read_gps_data(filename)
    [t, lon, lat, height] = textread(filename, '%f%f%f%f');
end
```

Listing: read_gps_data.m

```matlab
clear all, close all, clc;

gps_data = struct('time',   [], 'lon',  [], 'lat', [],...
                  'height', [], 'name', {''});

gps_data.name = 'BZ09';

[gps_data.time, gps_data.lon, gps_data.lat, ...
    gps_data.height ] = read_gps_data('BZ09.dat');

plot_gps_timeseries(gps_data);
```

Listing: plot_bz09.m

Intro – Do You Recognize This?

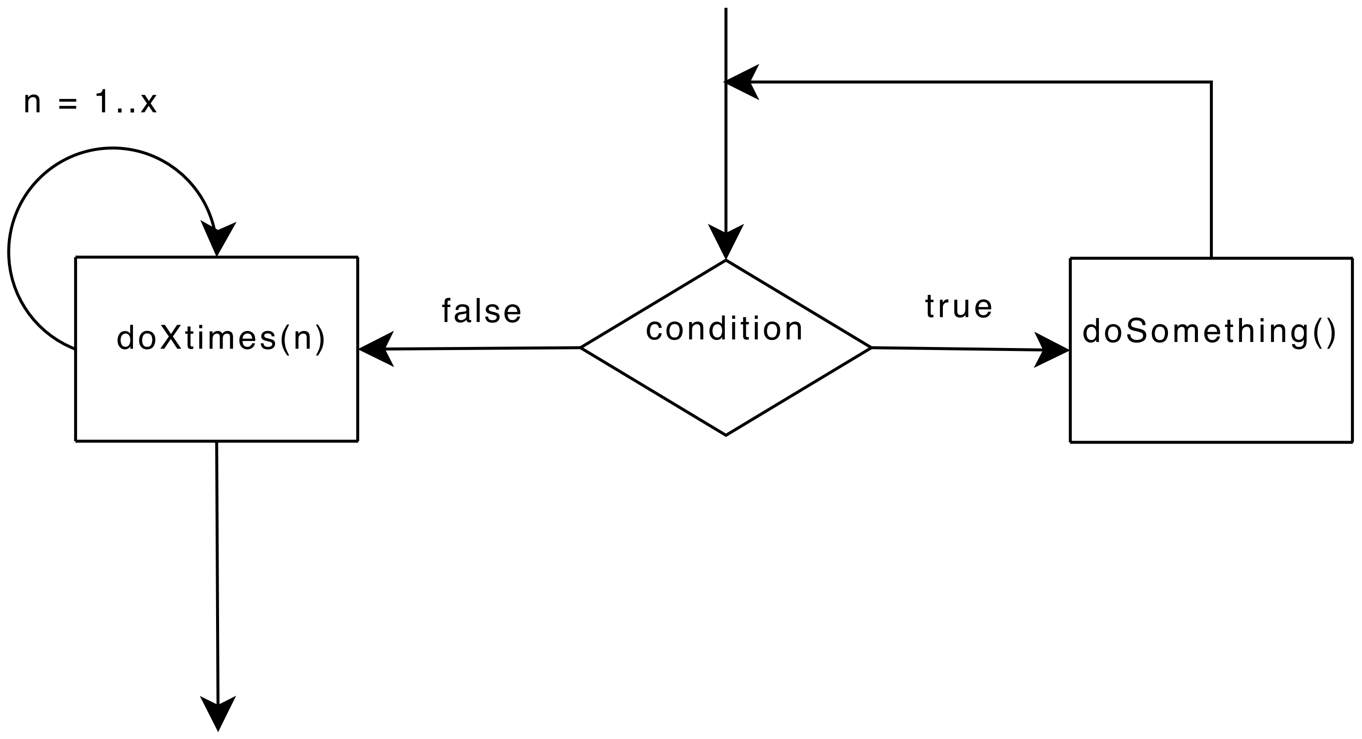
```
function plot_gps_timeseries(gps_struct)

    figure
    subplot(3,1,1)
    plot( gps_struct.time, gps_struct.lon-mean(gps_struct.lon) )
    title( sprintf('%s timeseries', gps_struct.name) )
    ylabel('lon (m)');

    subplot(3,1,2)
    plot( gps_struct.time, gps_struct.lat-mean(gps_struct.lat) )
    ylabel('lat (m)');

    subplot(3,1,3)
    plot( gps_struct.time, gps_struct.height-mean(gps_struct.height) )
    ylabel('height (m)');
    xlabel('epoch');
end
```

Listing: plot_gps_timeseries.m

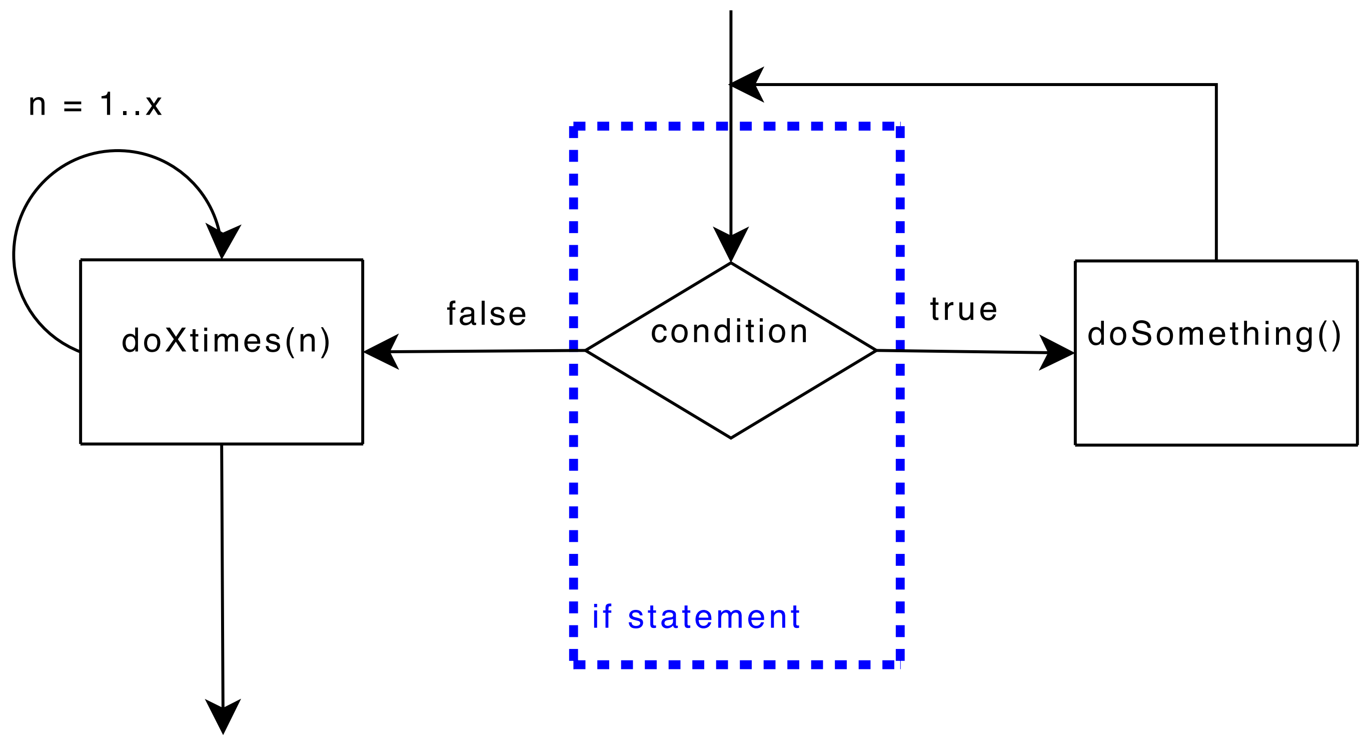
Today’s Schedule

- Truth Tables
- Control Structures
  - if – then – else
  - while
  - for
  - try – catch
  - If you heard about “`goto`”, forget about it
- A reminder of some good habits

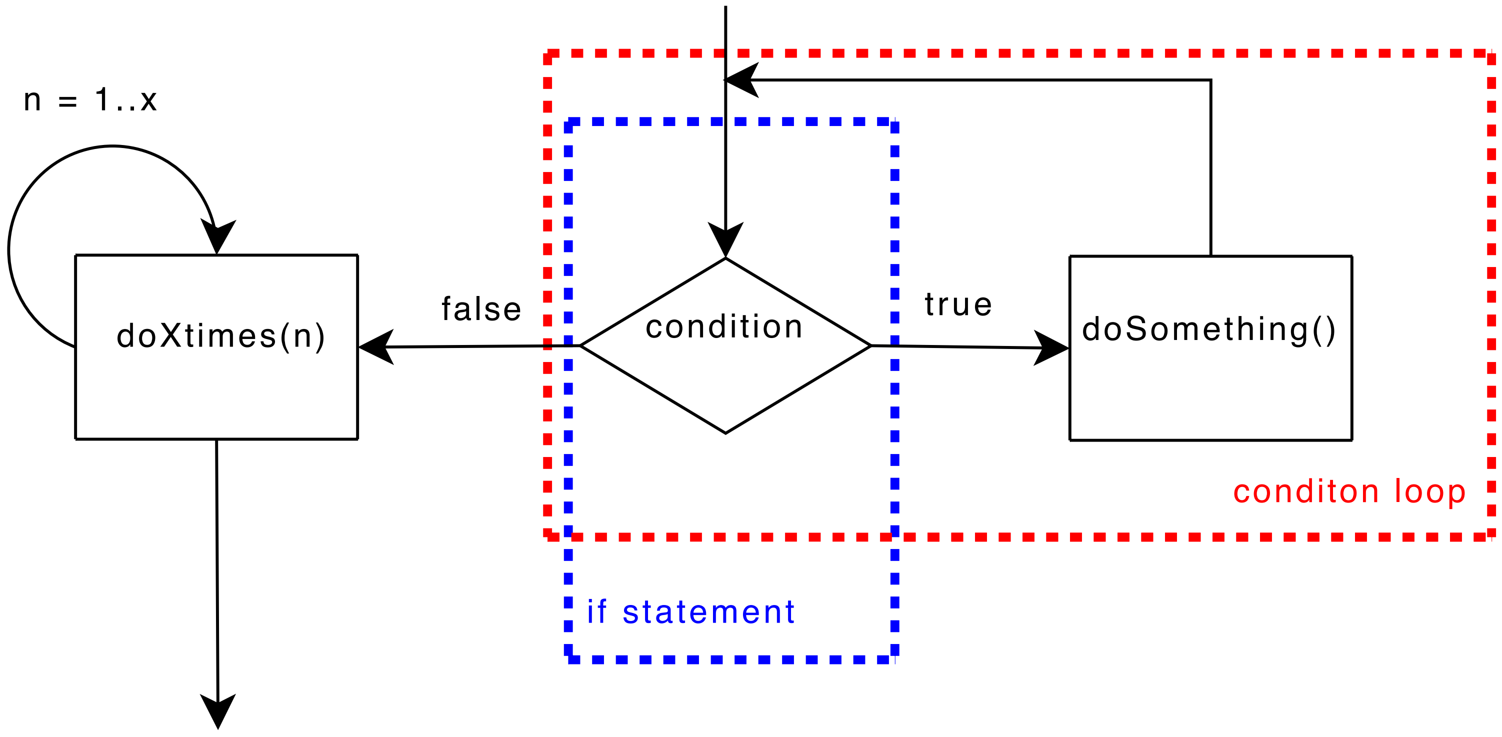
Control Flow: Redirecting the Stream



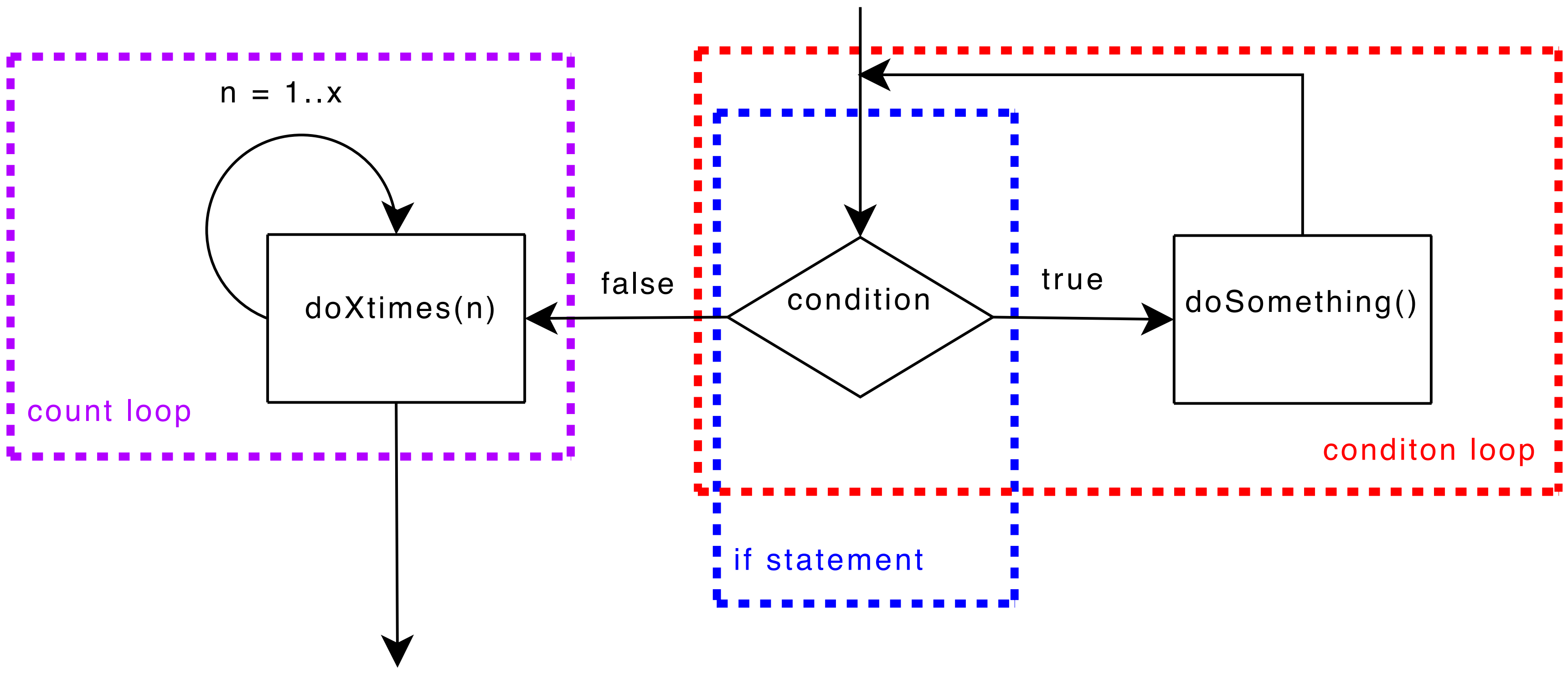
Control Flow: Redirecting the Stream



Control Flow: Redirecting the Stream



Control Flow: Redirecting the Stream



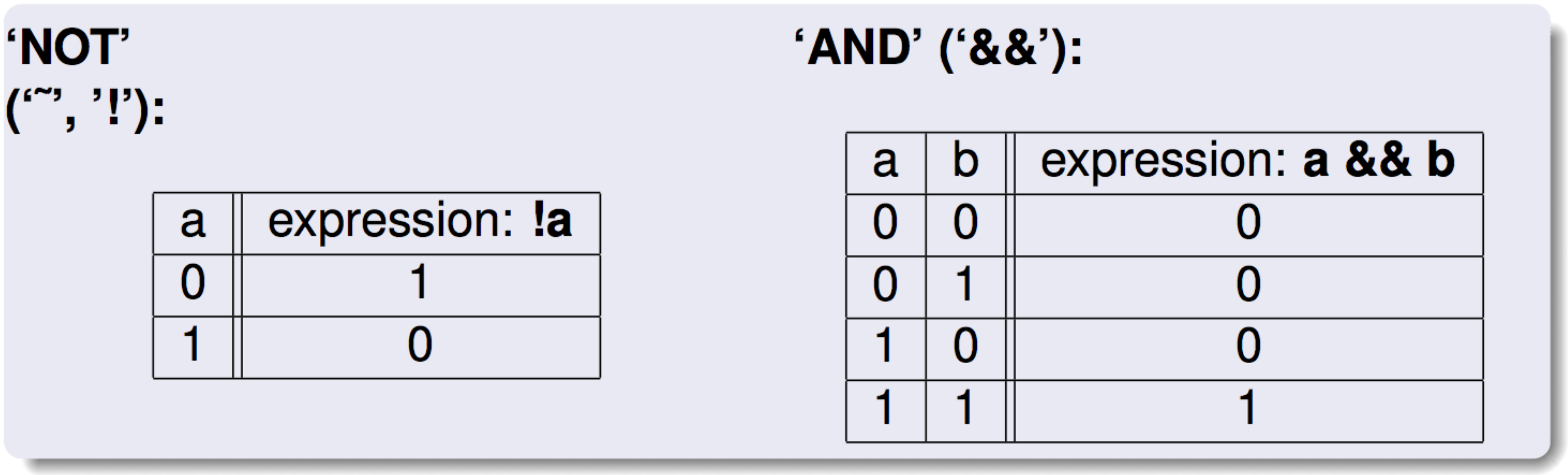
Flow Control Turns Batch Processing into Programming

- Allows different program behavior based on conditions you define – flow control
- A condition can be `true` (1) or `false` (0).
- You test a condition using the operators: <, <=, >, >=, ==,~= (for MATLAB)
- Functions often give numeric return values as answer to a test. In MATLAB, `strcmp('compare', 'strings')` will return 0 (false) if the strings are not the same.

Tests are Evaluated at Execution Time

- Most of the time, we will test the values of some variable, so that the program will do things differently according to the situation as it is run.
- Example questions to be tested:
  - Is the number of data points greater than 5?
  - Is the line number in this file divisible by 10?
  - Is today’s temperature a record low?

Truth Tables



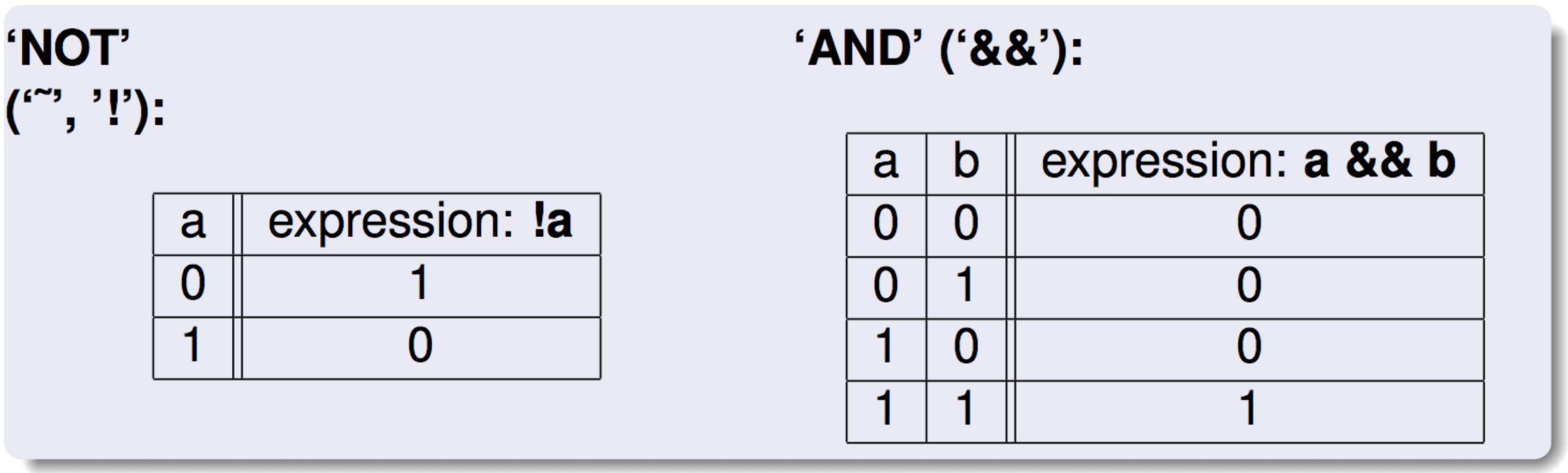
‘NOT’ (‘~’, ’!’):

a	expression: **!a**
0	1
1	0

‘AND’ (‘&&’):

a	b	expression: **a && b**
0	0	0
0	1	0
1	0	0
1	1	1

Truth Tables

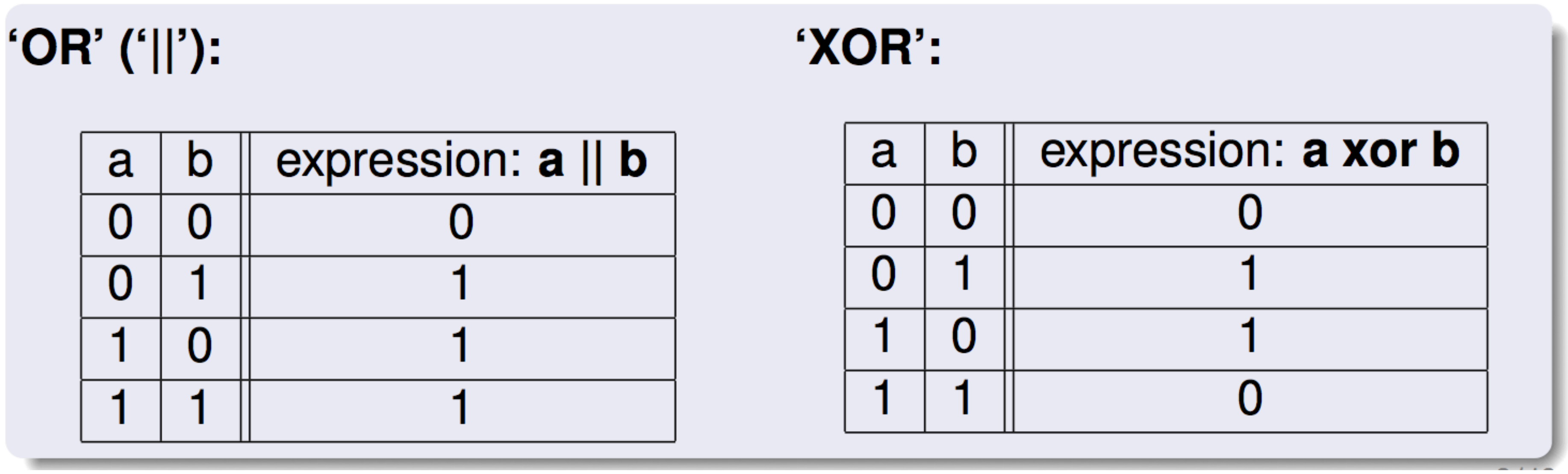


‘NOT’ (‘~’, ’!’):

a	expression: **!a**
0	1
1	0

‘AND’ (‘&&’):

a	b	expression: **a && b**
0	0	0
0	1	0
1	0	0
1	1	1



‘OR’ (‘||’):

a	b	expression: **a || b**
0	0	0
0	1	1
1	0	1
1	1	1

‘XOR’:

a	b	expression: **a xor b**
0	0	0
0	1	1
1	0	1
1	1	0

Check the Rules for Operator Precedence

- For MATLAB that is:
  1. Parentheses ( )
  2. Transpose (.'), power (.^), complex conjugate transpose('), matrix power(^)
  3. Unary plus (+), unary minus (−), logical negation (~)
  4. Multiplication(.*), right division(./), left division(.\), matrix multiplication (*), matrix right division (/), matrix left division (\)
  5. Addition(+), subtraction(−)
  6. Colon operator(:)
  7. Less than (<), less than or equal to (<=), greater than (>), greater than or equal to(>=), equal to(==), not equal to(~=)
  8. Element−wise AND (&)
  9. Element−wise OR (|)
  10. Short−circuit AND (&&)
  11. Short−circuit OR (||)

```
Hypoteneuse = sqrt(side1^2 + side2^2)
Z = height + -depth
amag = a'*a
```

Exercise

Exercise for you to work through:

a	b	c	(a && b) || c	(a || !b) && (a xor c)
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

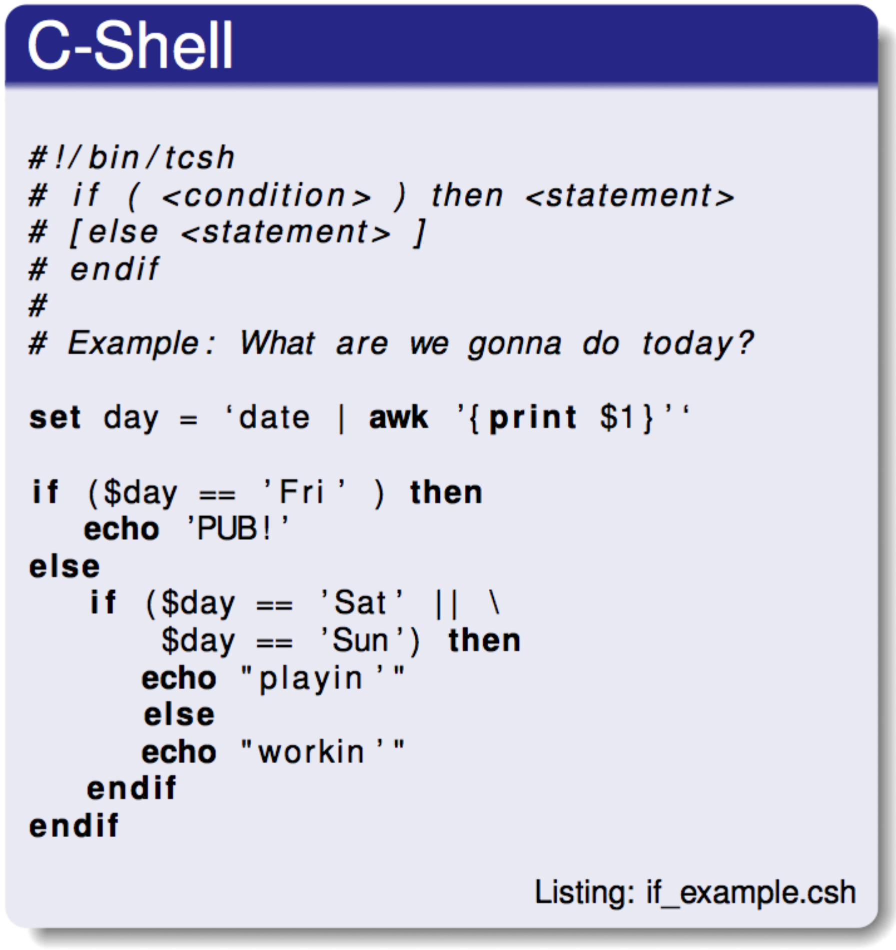
if – then – else

```
%
% if ( CONDITION ) STATEMENT
% [elseif ( CONDITION ) STATEMENT ]
% [else STATEMENT ]
% end.
%
% EXAMPLE: What are we going do today?
%

day=weekday(now);

if (day == 6 )
   disp('PUB!')
elseif (day == 1 || day == 7)
      disp('playing')
else
   disp('working')
end
```

- CONDITION
  - The condition must be true or false (1 or 0)
  - The condition can be an expression or a variable or a number
- STATEMENT
  - A single statement or a block of statements
- Optional parts
  - You don't have to use else or elseif

if – then – else

```
%
% if ( CONDITION ) STATEMENT
% [elseif ( CONDITION ) STATEMENT ]
% [else STATEMENT ]
% end.
%
% EXAMPLE: What are we going do today?
%

day=weekday(now);

if (day == 6 )
   disp('PUB!')
elseif (day == 1 || day == 7)
      disp('playing')
else
   disp('working')
end
```



C-Shell

```
#!/bin/tcsh
# if ( <condition> ) then <statement>
# [else <statement> ]
# endif
#
# Example: What are we gonna do today?

set day = ‘date | awk '{print $1}'‘

if ($day == 'Fri' ) then
   echo 'PUB!'
else
   if ($day == 'Sat' || \
       $day == 'Sun') then
      echo "playin'"
      else
      echo "workin'"
   endif
endif
```

Listing: if_example.csh

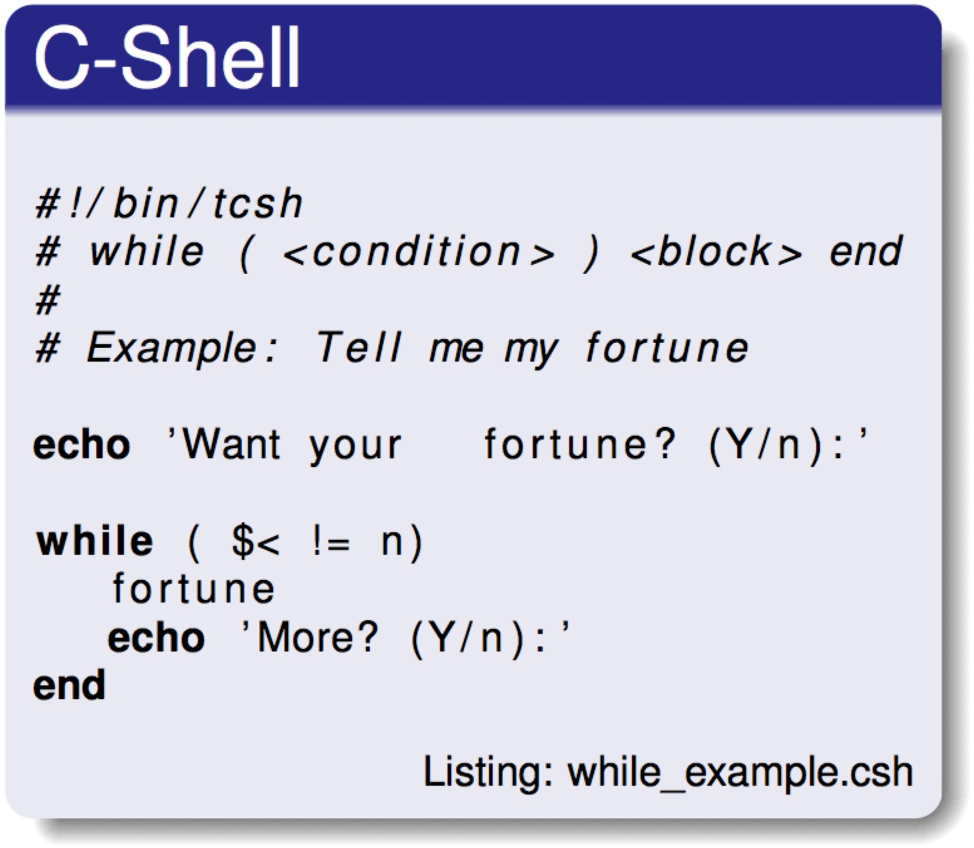
Condition-controlled loop (`while`)

```
% while ( CONDITION )
%   STATEMENT
% end.
%
% EXAMPLE: Read input until user has enough
%

clc;                %clear screen

n = 1;
disp('Read this text.')
string = ['You have read this ' ...
    num2str(n) ' times.'];
disp(string)
while (~strcmp( input('More? Y/n: ', 's'),'n'))
    n = n + 1;
    disp('Read this text.')
    string = ['You have read this ' ...
        num2str(n) ' times.'];
    disp(string)
end
```

- CONDITION
  - The condition must be true or false (1 or 0)
  - The condition can be an expression or a variable or a number
- STATEMENT
  - A single statement or a block of statements
- Optional parts
  - None

Condition-controlled loop (`while`)

```
% while ( CONDITION )
%   STATEMENT
% end.
%
% EXAMPLE: Read input until user has enough
%

clc;               %clear screen

n = 1;
disp('Read this text.')
string = ['You have read this ' ...
    num2str(n) ' times.'];
disp(string)
while (~strcmp( input('More? Y/n: ', 's'),'n'))
    n = n + 1;
    disp('Read this text.')
    string = ['You have read this ' ...
        num2str(n) ' times.'];
    disp(string)
end
```

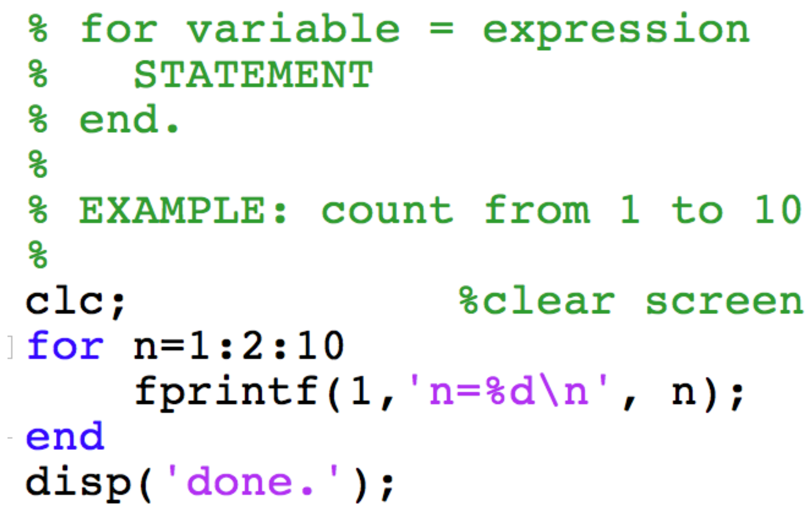


C-Shell

```
#!/bin/tcsh
# while ( <condition> ) <block> end
#
# Example: Tell me my fortune

echo 'Want your  fortune? (Y/n):'

while ( $< != n)
    fortune
    echo 'More? (Y/n):'
end
```

Listing: while_example.csh

Count-controlled loop (`for`)

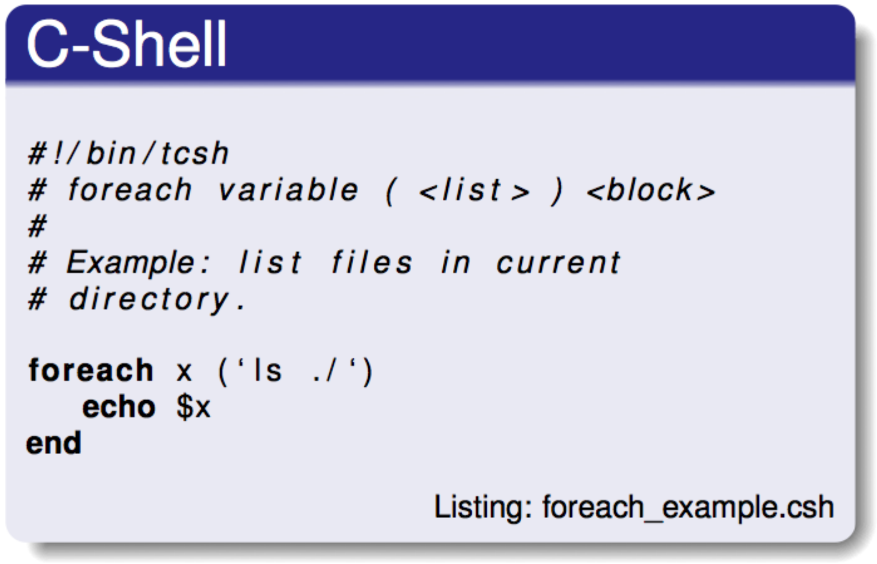


```
% for variable = expression
%    STATEMENT
% end.
%
% EXAMPLE: count from 1 to 10
%
clc;                   %clear screen
for n=1:2:10
    fprintf(1,'n=%d\n', n);
end
disp('done.');
```

- EXPRESSION
  - In MATLAB, the expression must produce an array of numbers
- STATEMENT
  - A single statement or a block of statements
- Optional parts
  - None

Count-controlled loop (`for`)

```
% for variable = expression
%   STATEMENT
% end.
%
% EXAMPLE: count from 1 to 10
%
clc;                %clear screen
for n=1:2:10
    fprintf(1,'n=%d\n', n);
end
disp('done.');
```



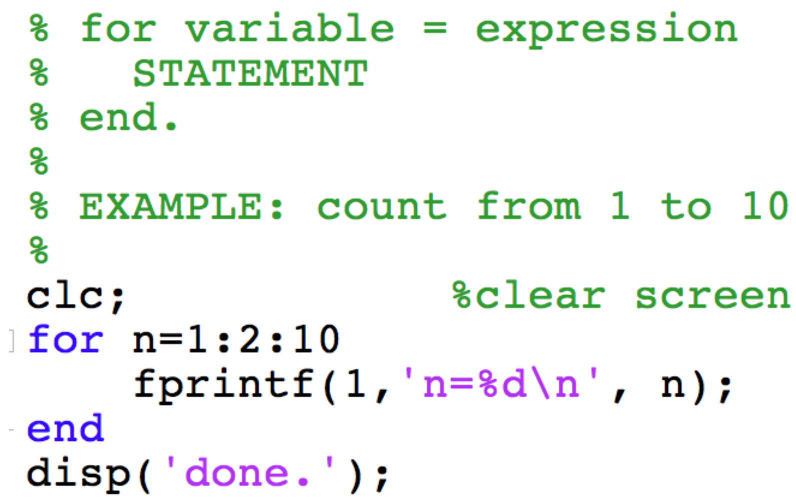
C-Shell

```
#!/bin/tcsh
# foreach variable ( <list> ) <block>
#
# Example: list files in current
# directory.

foreach x (‘ls ./‘)
    echo $x
end
```

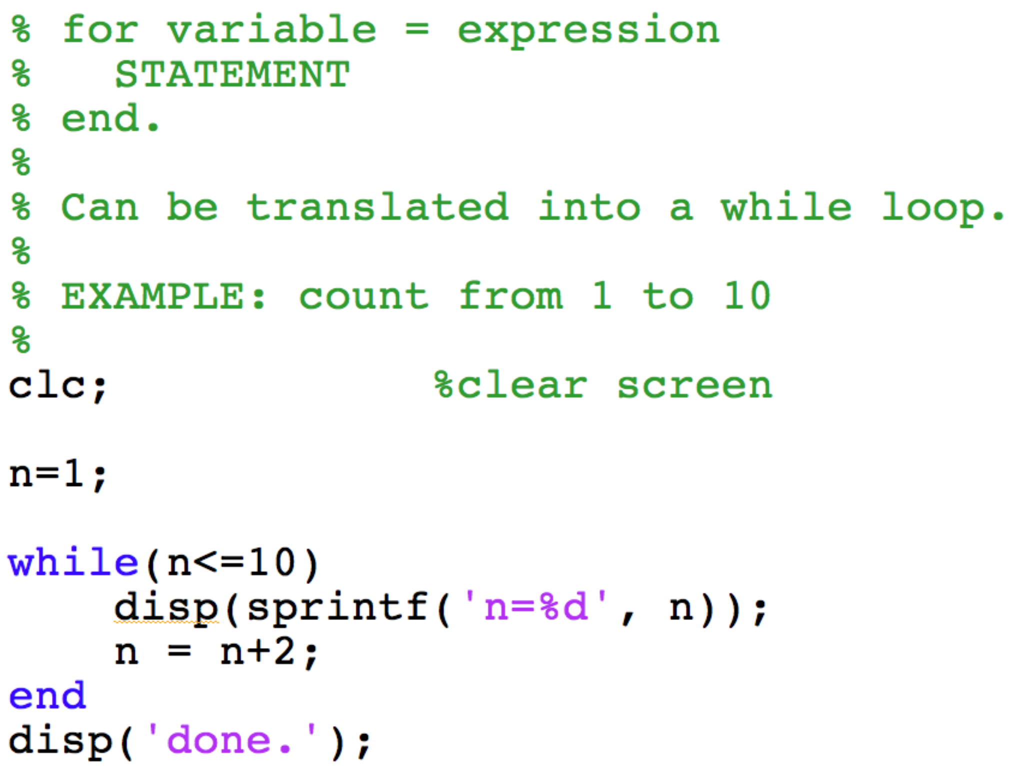
Listing: foreach_example.csh

Making a for loop using while

*You can exactly replicate a **for** loop using **while***



```
% for variable = expression
%   STATEMENT
% end.
%
% EXAMPLE: count from 1 to 10
%
clc;                %clear screen
for n=1:2:10
    fprintf(1,'n=%d\n', n);
end
disp('done.');
```



```
% for variable = expression
%   STATEMENT
% end.
%
% Can be translated into a while loop.
%
% EXAMPLE: count from 1 to 10
%
clc;                %clear screen

n=1;

while(n<=10)
    disp(sprintf('n=%d', n));
    n = n+2;
end
disp('done.');
```

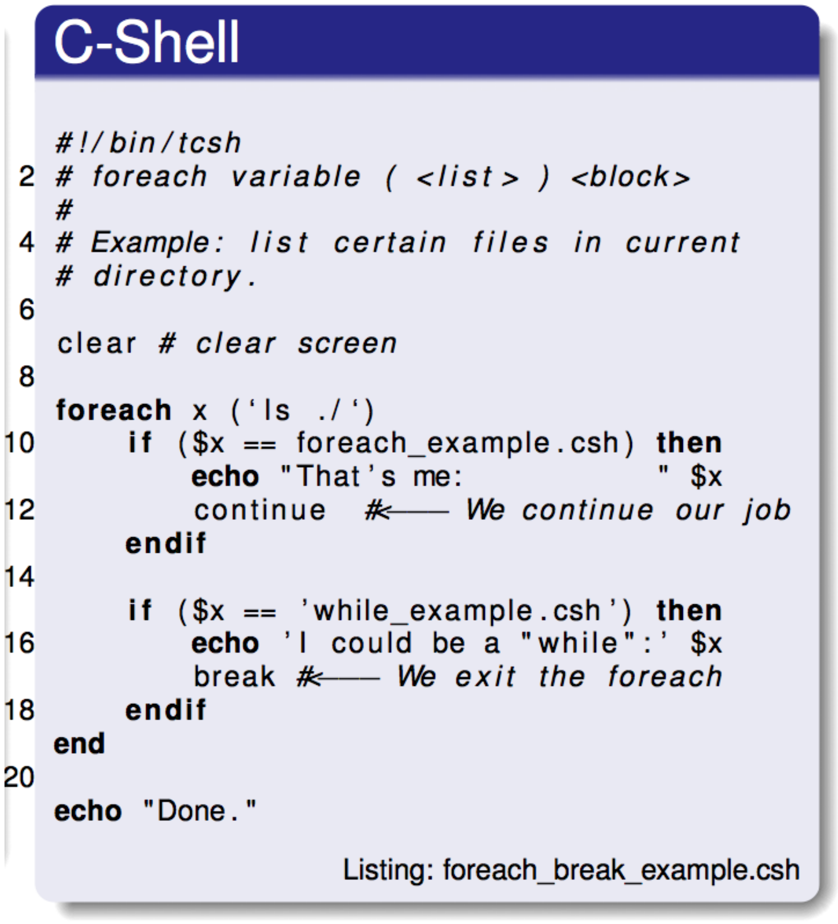
Breaking out of Loops

```
% for variable = expression
%    STATEMENT
% end.
%
% EXAMPLE: count from 1 to 10
%
clc;                %clear screen
for n=1:10
    if(n==2)
        disp(sprintf('TWO IS PRIME!'));
        continue;
    end
    if(n==5)
        disp( ... %note the dots !!!
           sprintf('Well, that''s enough!'));
        break;
    end
    disp(sprintf('n=%d', n));
end
disp('done.');
```

- CONTINUE
  - Skip to the end of the loop, and go around another time.
  - It is a way to avoid having to replicate a lot of code
- BREAK
  - Break completely out of the loop, and execute the statement after that.

Breaking out of Loops

```matlab
% for variable = expression
%   STATEMENT
% end.
%
% EXAMPLE: count from 1 to 10
%
clc;                %clear screen
for n=1:10
    if(n==2)
        disp(sprintf('TWO IS PRIME!'));
        continue;
    end
    if(n==5)
        disp( ... %note the dots !!!
           sprintf('Well, that''s enough!'));
        break;
    end
    disp(sprintf('n=%d', n));
end
disp('done.');
```



C-Shell

```csh
#!/bin/tcsh
# foreach variable ( <list> ) <block>
#
# Example: list certain files in current
# directory.

clear # clear screen

foreach x (`ls ./`)
    if ($x == foreach_example.csh) then
        echo "That's me:            " $x
        continue   #←— We continue our job
    endif

    if ($x == 'while_example.csh') then
        echo 'I could be a "while":' $x
        break #←— We exit the foreach
    endif
end

echo "Done."
```

Listing: foreach_break_example.csh

try-catch

- The ‘try-catch’ pair is a special type of flow control. It sets up a block of commands that will be executed, and then a set of commands to execute only if there is an error in the first block.
- This lets you deal with some kinds of errors (like missing files) more gracefully than by letting MATLAB beep and spew some red text at the user.

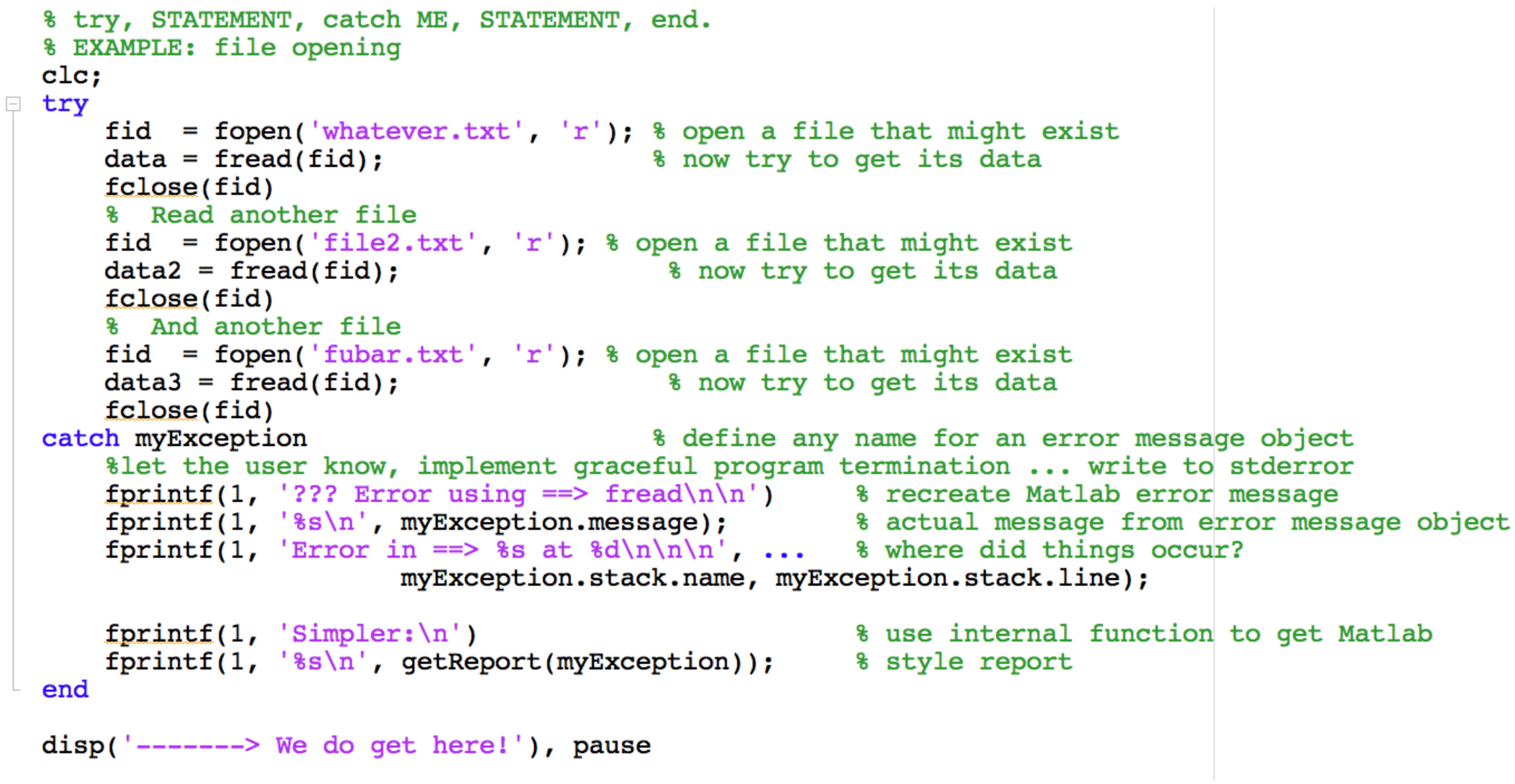
Error Control: `try-catch`

Matlab

```
% try , STATEMENT, catch ME, STATEMENT, end.
% EXAMPLE: file opening
clc;
try
    fid  = fopen('whatever.txt', 'r'); % open a non—existing file
    data = fread(fid);                 % now try to get its data
    fclose(fid)
catch myException                      % define any name for an error message object
    %let the user know, implement graceful program termination ... write to stderror
    fprintf(2, '??? Error using ==> fread\n\n')     % recreate Matlab error message
    fprintf(2, '%s\n', myException.message);        % actual message from error message object
    fprintf(2, 'Error in ==> %s at %d\n\n\n', ...   % where did things occur?
               myException.stack.name, myException.stack.line);

    fprintf(1, 'Simpler:\n')                        % use internal function to get Matlab
    fprintf(2, '%s\n', getReport(myException));     % style report
end

disp('———————> We do get here!'), pause

%now without try—catch ...
fid  = fopen('whatever.txt', 'r');
data = fread(fid);

disp('We cannot get here!')            % We'll only make it here if 'whatever.txt' exists!
```

Listing: try_catch_example.m

Error Control: `try-catch`



```
% try, STATEMENT, catch ME, STATEMENT, end.
% EXAMPLE: file opening
clc;
try
    fid  = fopen('whatever.txt', 'r'); % open a file that might exist
    data = fread(fid);                 % now try to get its data
    fclose(fid)
    %  Read another file
    fid  = fopen('file2.txt', 'r'); % open a file that might exist
    data2 = fread(fid);                 % now try to get its data
    fclose(fid)
    %  And another file
    fid  = fopen('fubar.txt', 'r'); % open a file that might exist
    data3 = fread(fid);                 % now try to get its data
    fclose(fid)
catch myException                      % define any name for an error message object
    %let the user know, implement graceful program termination ... write to stderror
    fprintf(1, '??? Error using ==> fread\n\n')     % recreate Matlab error message
    fprintf(1, '%s\n', myException.message);        % actual message from error message object
    fprintf(1, 'Error in ==> %s at %d\n\n\n', ...   % where did things occur?
                       myException.stack.name, myException.stack.line);

    fprintf(1, 'Simpler:\n')                        % use internal function to get Matlab
    fprintf(1, '%s\n', getReport(myException));     % style report
end

disp('-------> We do get here!'), pause
```

Don’t Even Think of Using `goto`

I COULD RESTRUCTURE THE PROGRAM'S FLOW
OR USE ONE LITTLE 'GOTO' INSTEAD.
EH, SCREW GOOD PRACTICE. HOW BAD CAN IT BE?
goto main_sub3;
COMPILE

“GOTO”, http://xkcd.com/292

Some Comments and General Advice

- You don't have to start with an empty file – that's intimidating: use old file as 'template'
- Put it in a script, unless it's a (short) one-liner (you won't use again).
- Make it a habit to include `clear all, close all, clc;` at the beginning of your scripts
- Keep things nice and clean: definition of function in function file; use of function on command line or in script file

More good practice

- Use indentations to structure your code (align comments etc)
- Use meaningful variable and function names (sec instead of i and listFiles() instead of lfls())
- Decide for one formatting and naming scheme and stick to it; no matter which one it is.
- Comment your code
- Do not over comment your code!
- Try and catch errors
- Self-study: http://www.google.com/search?hl=en&q=good+programming+style&btnG=Search