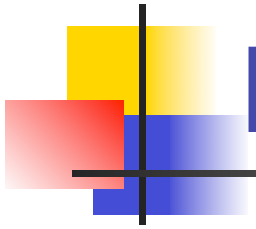
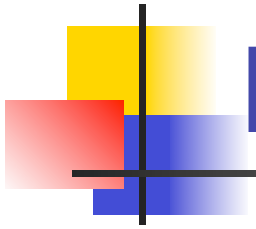


ECOR 1010

Lecture 17

MATLAB Programming

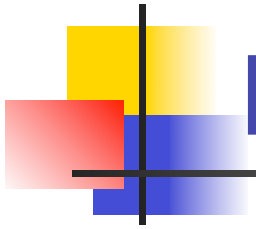


MATLAB Programming

- Use MATLAB to solve programming related problems
- Simple and practical programming language
- Generally use the Editor window
- Can develop script m-files and function m-files.

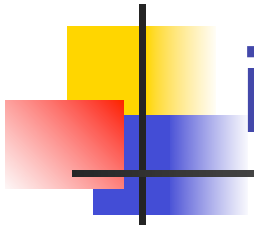
Relational and Logical Operators

<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
~=	Not equal to
~	Not
&	And
|	Or



Relational and Logical Operators

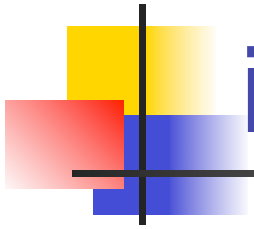
- Useful when comparing and selecting values
- Used to determine if an expression will evaluate to true or false.
- Used in if statements and in while loops.
- Used whenever a logical decision must be made



if statements

```
if (expression)
    statements
end
```

- If the expression evaluates to true, then the statements between the if command and the end command are executed.
- If the logical expression evaluates to false, then the program will skip the actions inside of the if statement and jump to the statement immediately after the end statement.

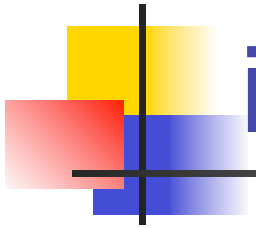


if statements

```
if expression
   statements
end
```

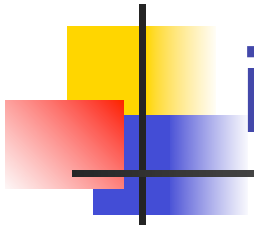
- Any number of commands can be included in the statements section, and the semicolon rules still apply to these commands.

```
if expression
   statement1;
   statement2;
   statement3
end
```



if statements

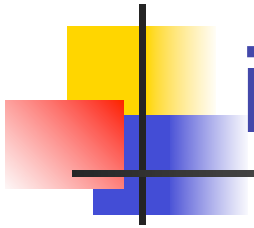
- Create an if statement that will write a message if a number is greater than 6.



if statements

- Create an if statement that will write a message if a number is greater than 6.

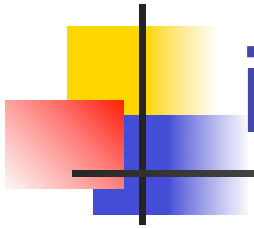
```
Number = 8; %this value can be changed
```



if statements

- Create an if statement that will write a message if a number is greater than 6.

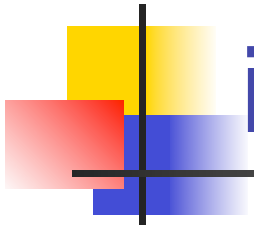
```
Number = 8; %this value can be changed
if Number > 6
```



if statements

- Create an if statement that will write a message if a number is greater than 6.

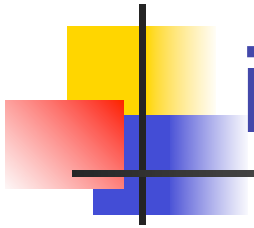
```
Number = 8; %this value can be changed
if Number > 6
   disp('They''re learning for free!')
```



if statements

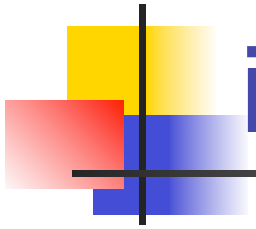
- Create an if statement that will write a message if a number is greater than 6.

```
Number = 8; %this value can be changed
if Number > 6
   disp('They''re learning for free!')
end
```



if statements

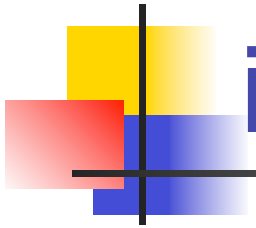
- Create an if statement that will test to see if a student is old enough to go to the bar (Must be at least 19). If they are, display an appropriate message.



if statements

- Create an if statement that will test to see if a student is old enough to go to the bar (Must be at least 19). If they are, display an appropriate message.

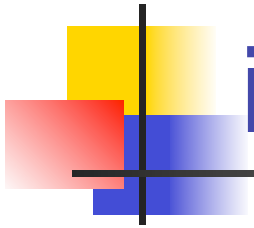
```
Age = 18; %this can be changed
```



if statements

- Create an if statement that will test to see if a student is old enough to go to the bar (Must be at least 19). If they are, display an appropriate message.

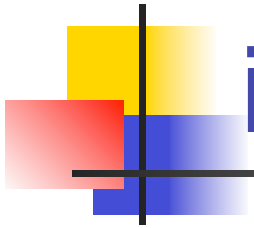
```
Age = 18; %this can be changed
if Age >= 19
```



if statements

- Create an if statement that will test to see if a student is old enough to go to the bar (Must be at least 19). If they are, display an appropriate message.

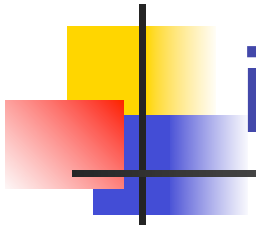
```
Age = 18; %this can be changed
if Age >= 19
    disp('Old enough to drink')
```



if statements

- Create an if statement that will test to see if a student is old enough to go to the bar (Must be at least 19). If they are, display an appropriate message.

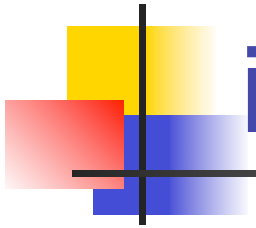
```
Age = 18; %this can be changed
if Age >= 19
    disp('Old enough to drink')
end
```



if-else Statements

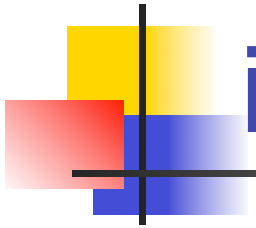
```
if expression
    statements
else
    statements
end
```

- Very similar to an if statement
- If the expression is true, the first set of statements are performed.
- If the expression evaluates to false, the second set (else) of statements are performed.
- An action is performed in either case



if-else Statements

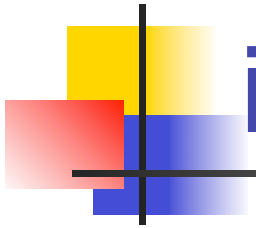
- Write an if statement that will test to see if a grade is an A (85.0-89.9). If the grade is an A, then increase a counter, assign the appropriate number of grade points to a variable Grade_Points, and display an appropriate message. If the grade is not an A, display an appropriate message.



if-else Statements

- Write an if statement that will test to see if a grade is an A (85.0-89.9). If the grade is an A, then increase a counter, assign the appropriate number of grade points to a variable Grade_Points, and display an appropriate message. If the grade is not an A, display an appropriate message.

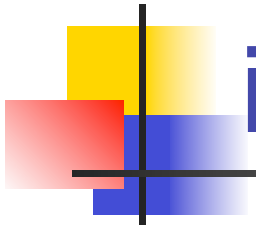
```
Grade = 86.3 %can be changed
```



if-else Statements

- Write an if statement that will test to see if a grade is an A (85.0-89.9). If the grade is an A, then increase a counter, assign the appropriate number of grade points to a variable Grade_Points, and display an appropriate message. If the grade is not an A, display an appropriate message.

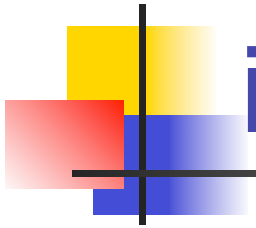
```
Grade = 86.3 %can be changed
if (Grade >= 85.0) & (Grade < 90.0)
```



if-else Statements

- Write an if statement that will test to see if a grade is an A (85.0-89.9). If the grade is an A, then increase a counter, assign the appropriate number of grade points to a variable Grade_Points, and display an appropriate message. If the grade is not an A, display an appropriate message.

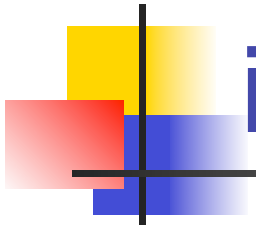
```
Grade = 86.3 %can be changed
if (Grade >= 85.0) & (Grade < 90.0)
   Grade_Points = 11;
```



if-else Statements

- Write an if statement that will test to see if a grade is an A (85.0-89.9). If the grade is an A, then increase a counter, assign the appropriate number of grade points to a variable Grade_Points, and display an appropriate message. If the grade is not an A, display an appropriate message.

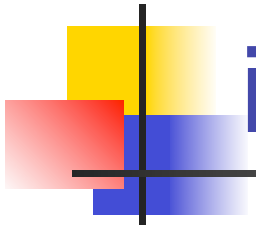
```
Grade = 86.3 %can be changed
if (Grade >= 85.0) & (Grade < 90.0)
   Grade_Points = 11;
   Counter = Counter + 1;
```



if-else Statements

- Write an if statement that will test to see if a grade is an A (85.0-89.9). If the grade is an A, then increase a counter, assign the appropriate number of grade points to a variable Grade_Points, and display an appropriate message. If the grade is not an A, display an appropriate message.

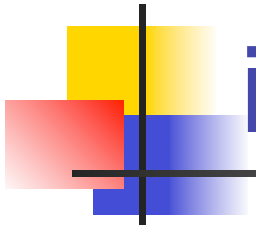
```
Grade = 86.3 %can be changed
if (Grade >= 85.0) & (Grade < 90.0)
   Grade_Points = 11;
   Counter = Counter + 1;
   disp('The grade is an A')
```



if-else Statements

- Write an if statement that will test to see if a grade is an A (85.0-89.9). If the grade is an A, then increase a counter, assign the appropriate number of grade points to a variable Grade_Points, and display an appropriate message. If the grade is not an A, display an appropriate message.

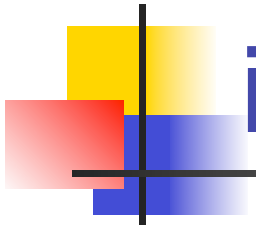
```
Grade = 86.3 %can be changed
if (Grade >= 85.0) & (Grade < 90.0)
   Grade_Points = 11;
   Counter = Counter + 1;
   disp('The grade is an A')
else
```



if-else Statements

- Write an if statement that will test to see if a grade is an A (85.0-89.9). If the grade is an A, then increase a counter, assign the appropriate number of grade points to a variable Grade_Points, and display an appropriate message. If the grade is not an A, display an appropriate message.

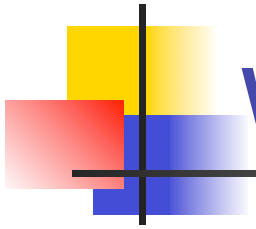
```
Grade = 86.3 %can be changed
if (Grade >= 85.0) & (Grade < 90.0)
   Grade_Points = 11;
   Counter = Counter + 1;
   disp('The grade is an A')
else
   disp('The grade is not an A')
```



if-else Statements

- Write an if statement that will test to see if a grade is an A (85.0-89.9). If the grade is an A, then increase a counter, assign the appropriate number of grade points to a variable Grade_Points, and display an appropriate message. If the grade is not an A, display an appropriate message.

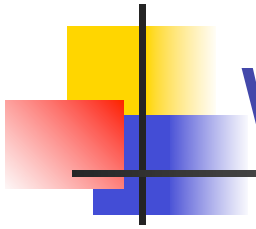
```
Grade = 86.3 %can be changed
if (Grade >= 85.0) & (Grade < 90.0)
   Grade_Points = 11;
   Counter = Counter + 1;
   disp('The grade is an A')
else
   disp('The grade is not an A')
end
```



while Loops

```
while expression
    statements
end
```

- Used to repeat a set of commands as long as the specified condition continues to evaluate to true
- Tests the expression before the actions are performed.
- If the expression evaluates to false, no actions within the loop will be performed.

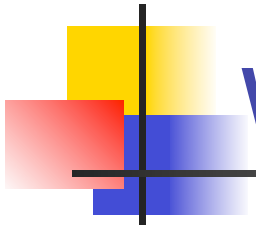


while Loops

```
while expression
   statements
end
```

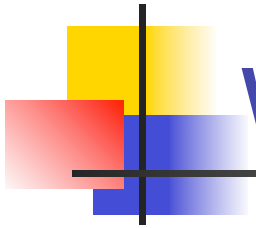
- The end command signifies the end of the loop. The program will then go back to the beginning of the loop to re-evaluate the expression to decide if the loop will be performed again.
- You can have any number of commands inside the loop:

```
while expression
   statement1
   statement2;
   statement3
end
```



while Loops

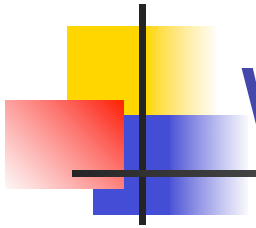
- Find the first positive even integer whose square is greater than, or equal to 2000



while Loops

- Find the first positive even integer whose square is greater than, or equal to 2000

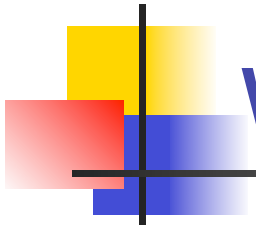
```
Number = 2;
```



while Loops

- Find the first positive even integer whose square is greater than, or equal to 2000

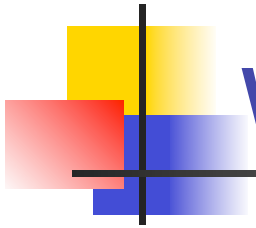
```
Number = 2;

Square = Number * Number;
```



while Loops

- Find the first positive even integer whose square is greater than, or equal to 2000

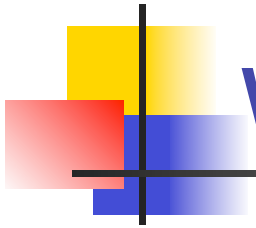
```
Number = 2;
Square = Number * Number;


while Square < 2000
```



while Loops

- Find the first positive even integer whose square is greater than, or equal to 2000

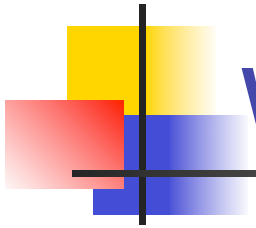
```
Number = 2;
Square = Number * Number;

while Square < 2000
    Number = Number + 2;
```



while Loops

- Find the first positive even integer whose square is greater than, or equal to 2000

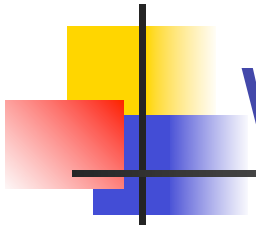
```
Number = 2;
Square = Number * Number;

while Square < 2000
    Number = Number + 2;
    Square = Number * Number;
```



while Loops

- Find the first positive even integer whose square is greater than, or equal to 2000

```
Number = 2;
Square = Number * Number;

while Square < 2000
    Number = Number + 2;
    Square = Number * Number;
end
```



while Loops

- Find the first positive even integer whose square is greater than, or equal to 2000

```
Number = 2;
Square = Number * Number;

while Square < 2000
    Number = Number + 2;
    Square = Number * Number;
end

disp(Number)
```

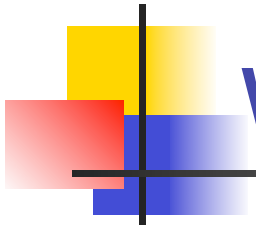
while Loops

- Find the first positive even integer whose square is greater than, or equal to 2000

```
Number = 2;
Square = Number * Number;

while Square < 2000
    Number = Number + 2;
    Square = Number * Number;
end

disp(Number)
```

Solution:

» 46

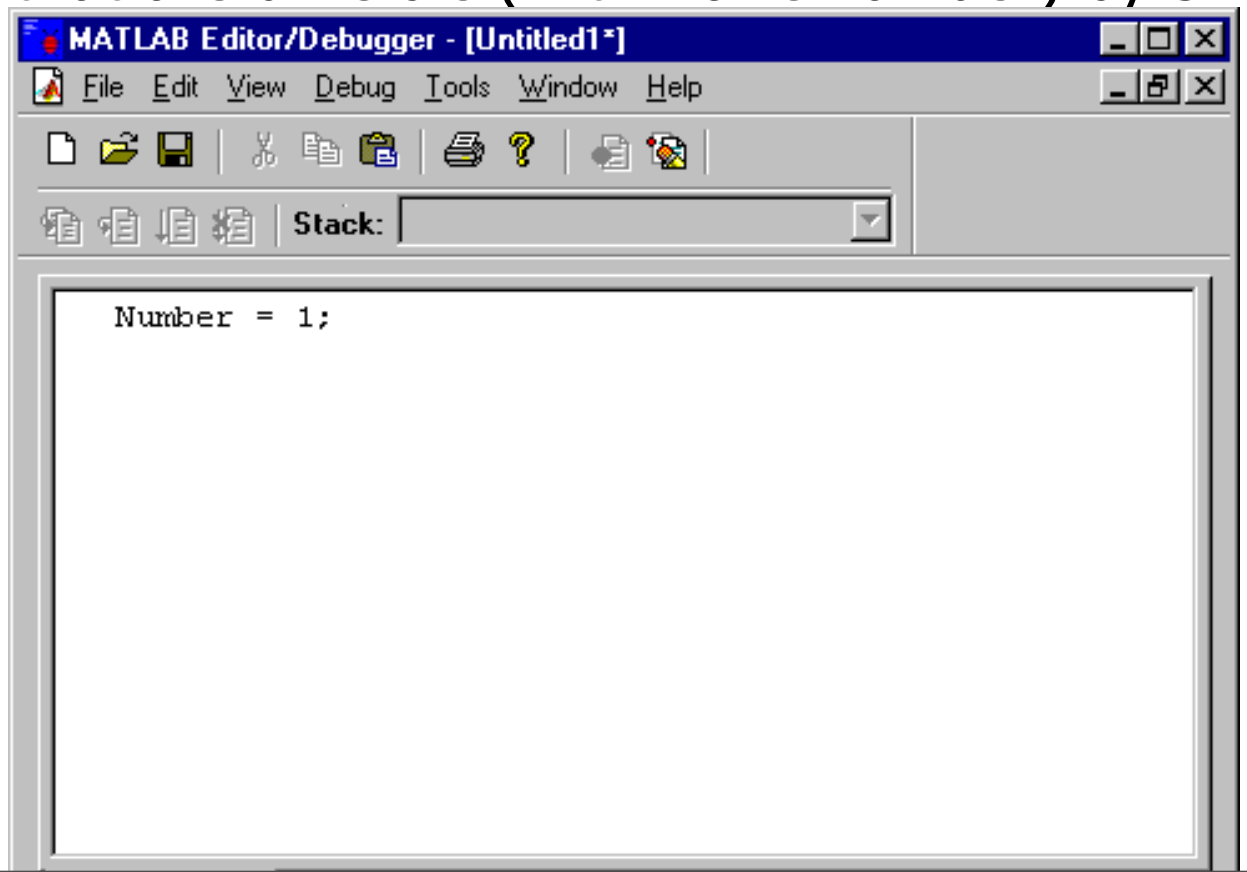


while Loops

- Count the number of integers between 1 and 986 that are divisible (with no remainder) by 3.

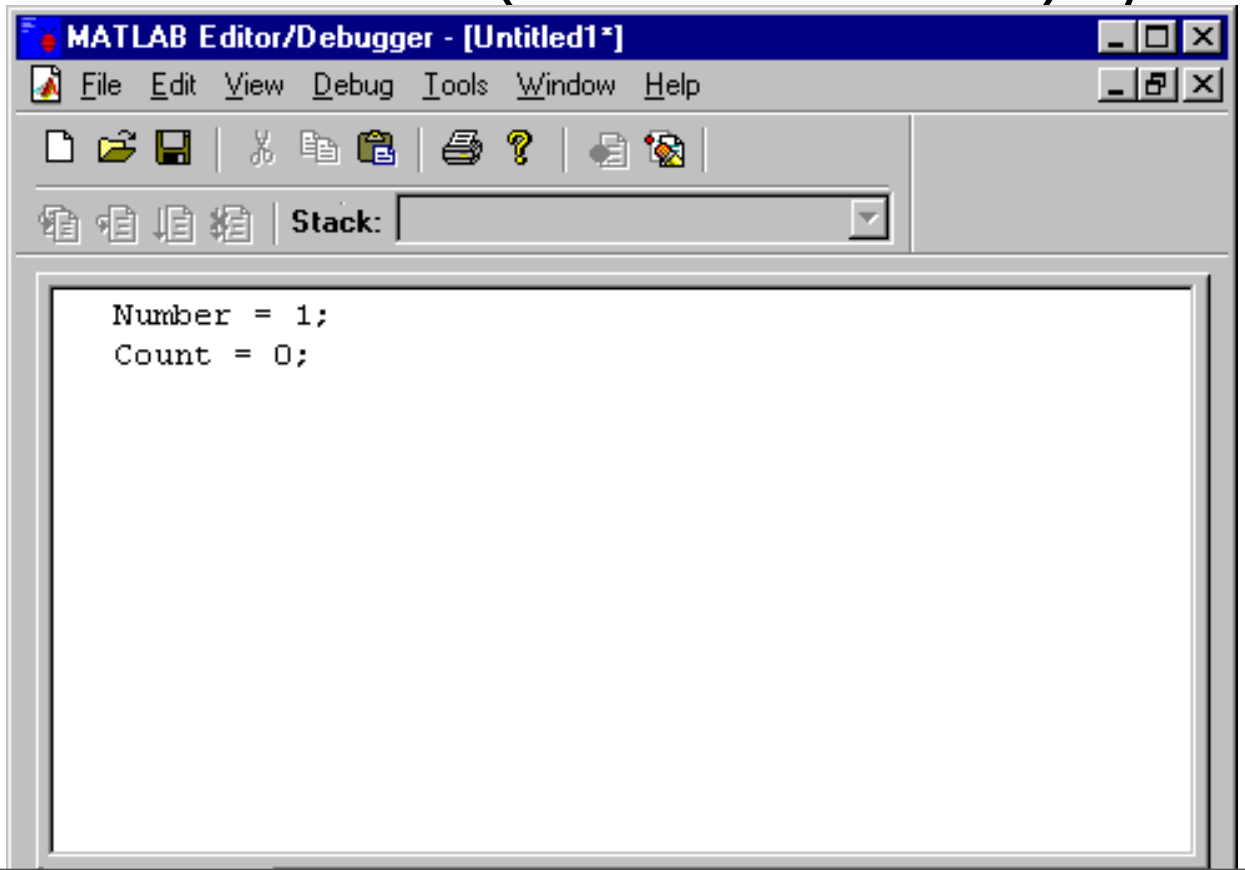
while Loops

- Count the number of integers between 1 and 986 that are divisible (with no remainder) by 3.



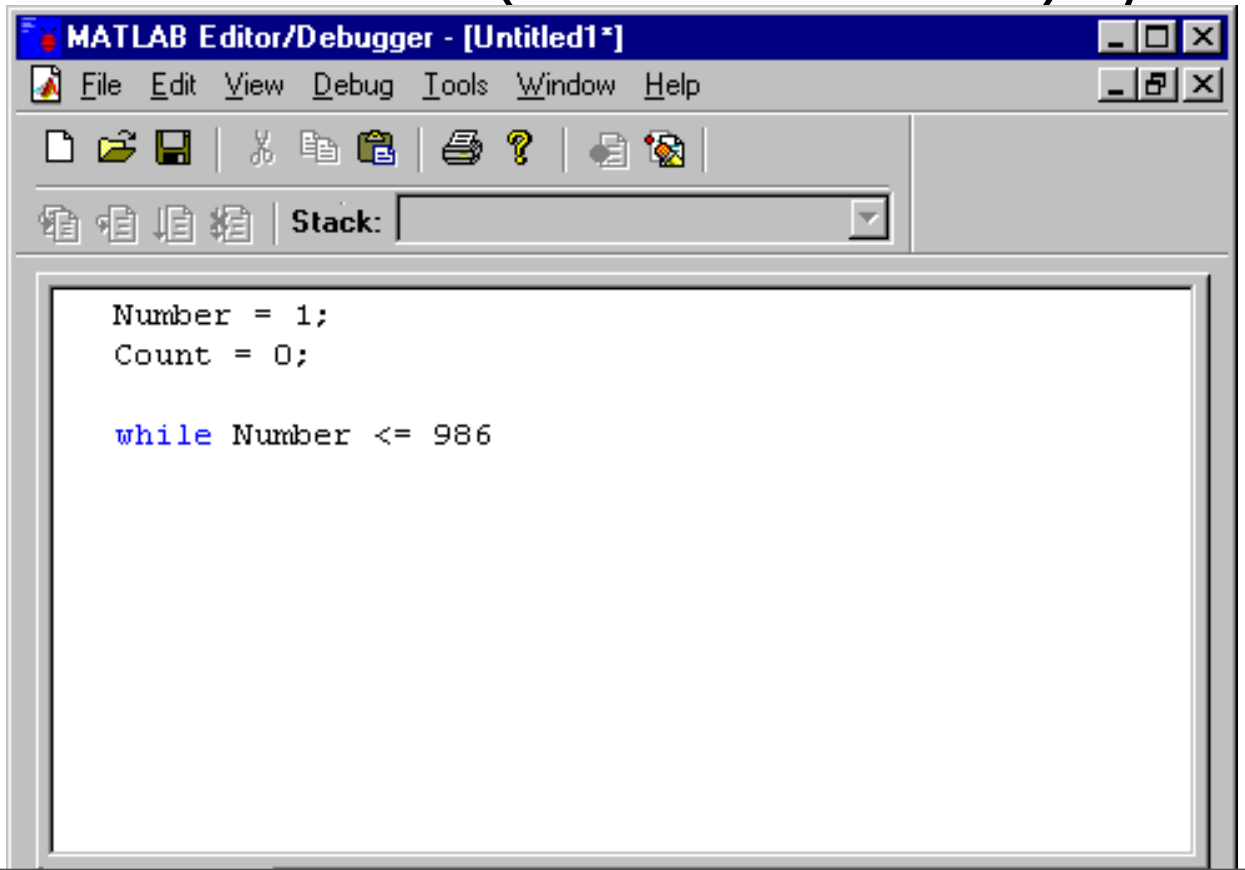
while Loops

- Count the number of integers between 1 and 986 that are divisible (with no remainder) by 3.



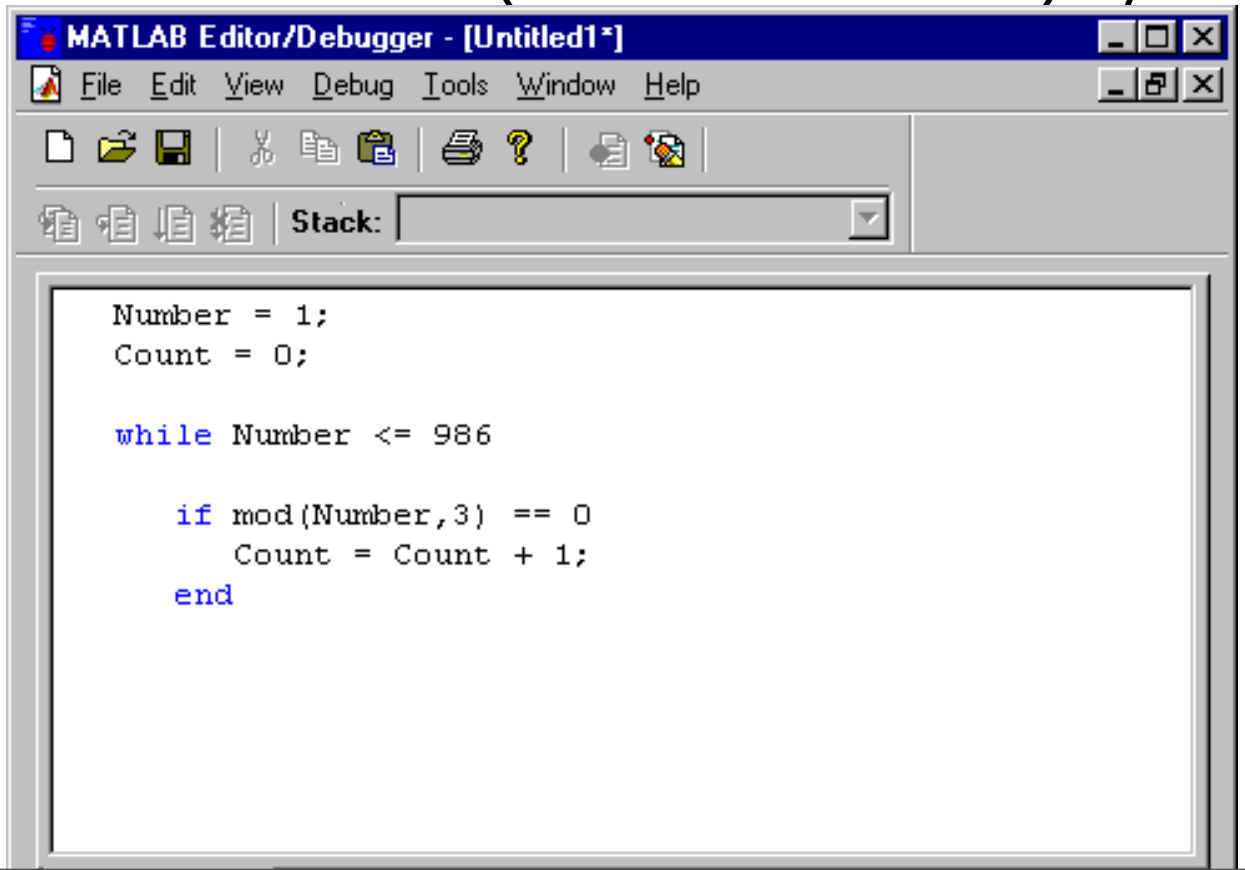
while Loops

- Count the number of integers between 1 and 986 that are divisible (with no remainder) by 3.



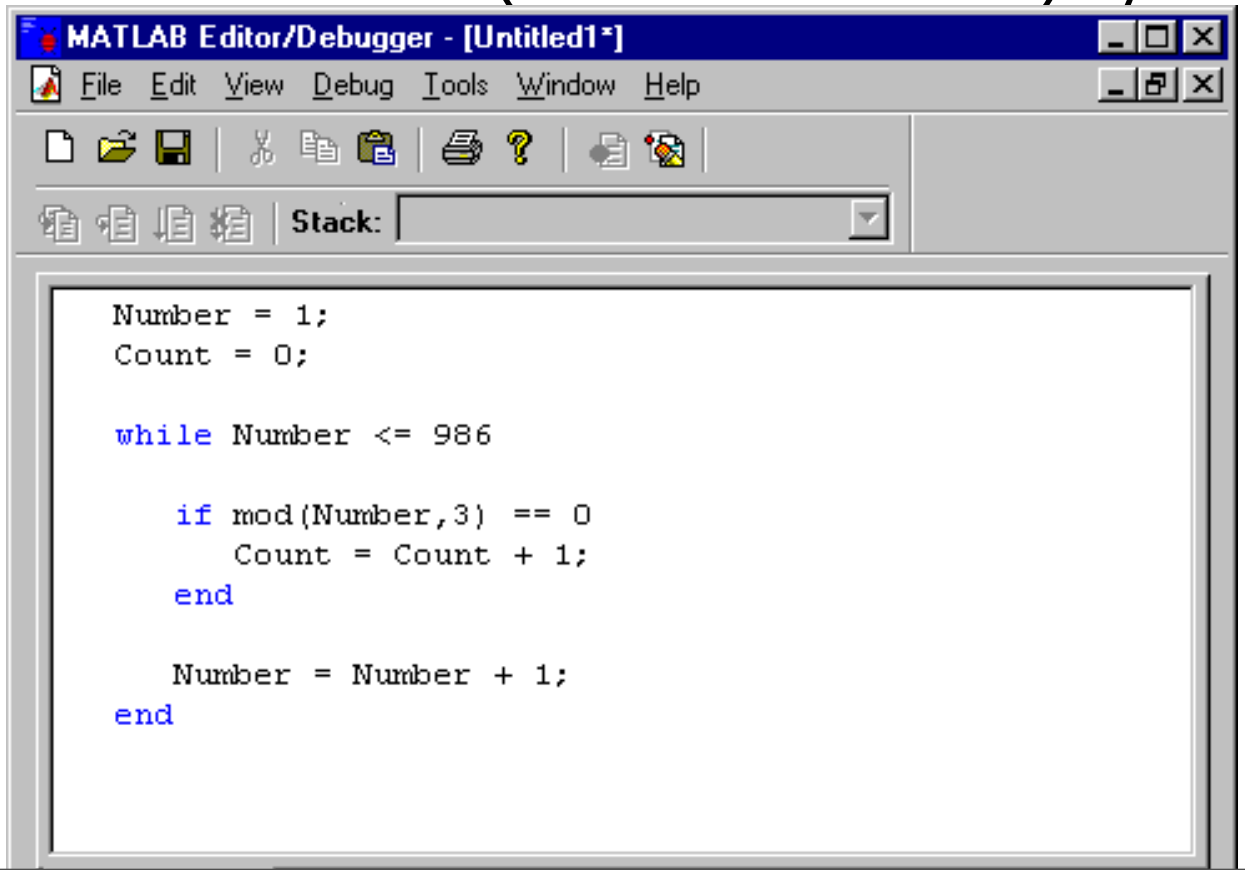
while Loops

- Count the number of integers between 1 and 986 that are divisible (with no remainder) by 3.



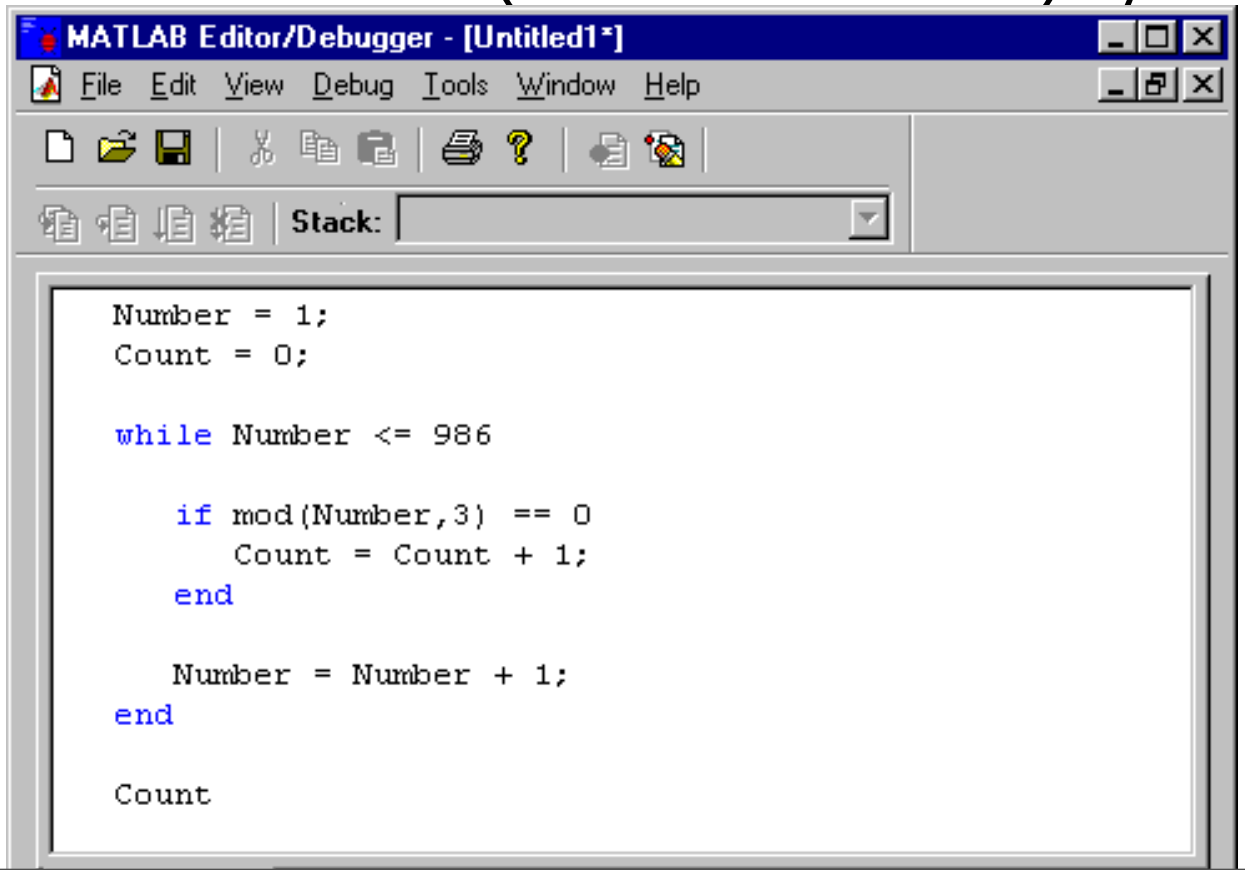
while Loops

- Count the number of integers between 1 and 986 that are divisible (with no remainder) by 3.



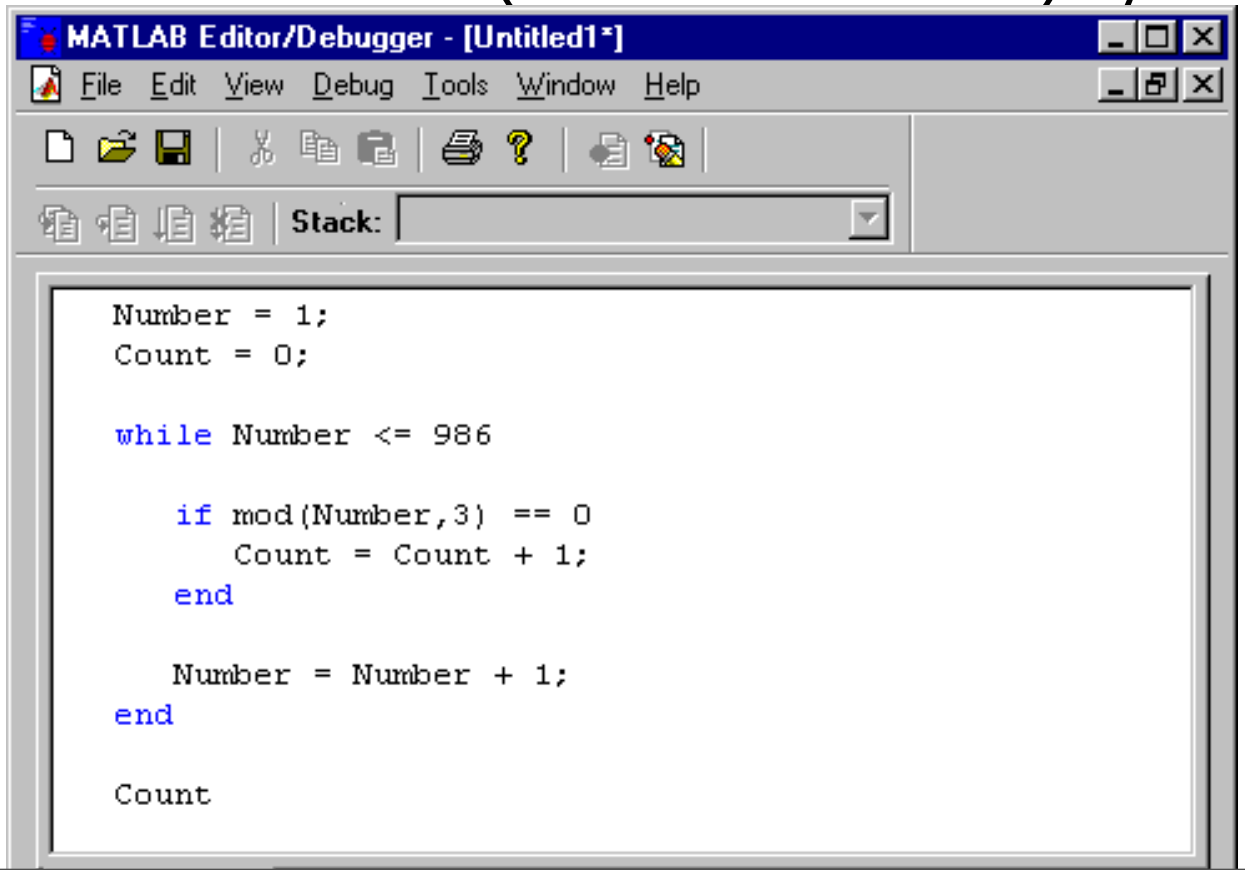
while Loops

- Count the number of integers between 1 and 986 that are divisible (with no remainder) by 3.



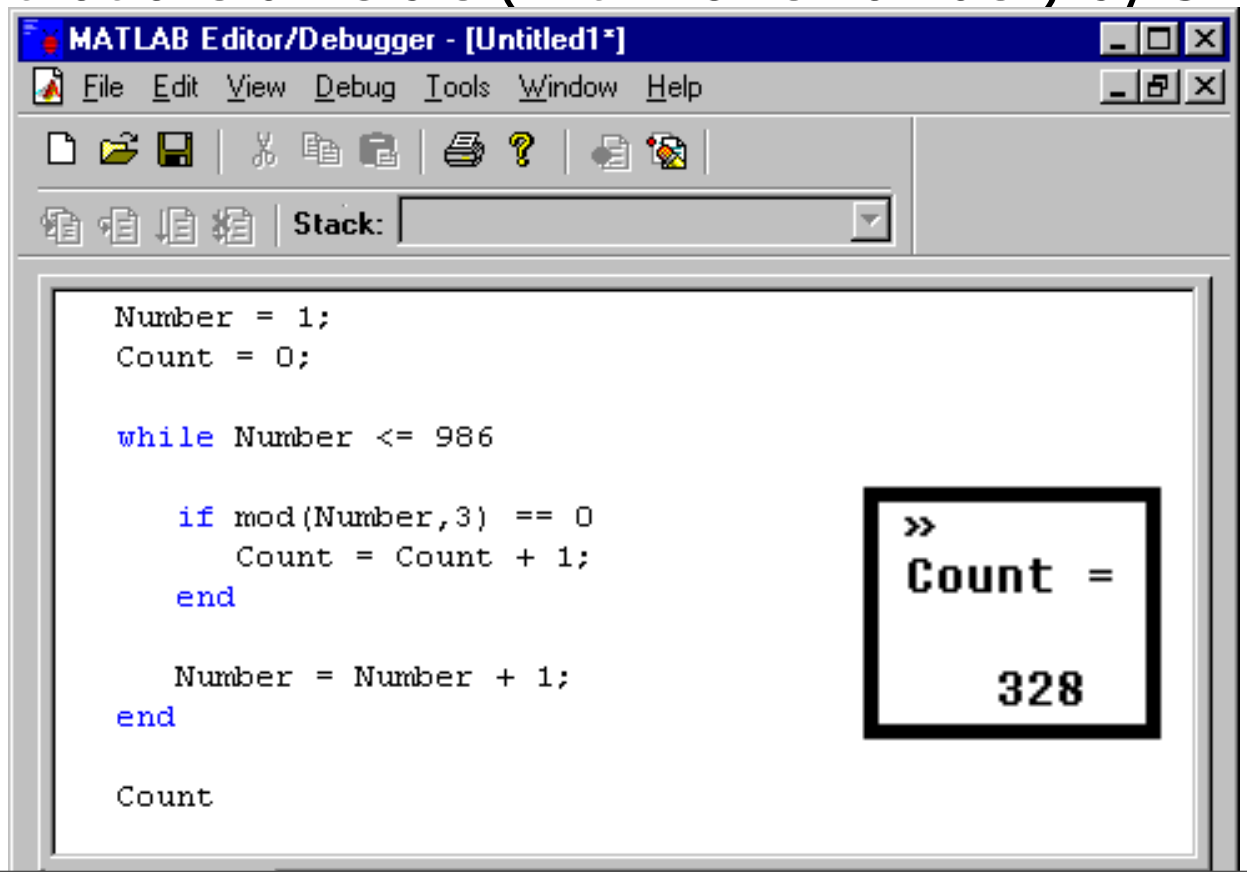
while Loops

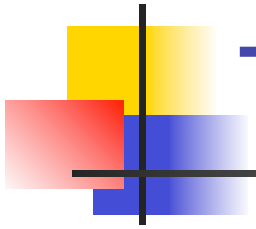
- Count the number of integers between 1 and 986 that are divisible (with no remainder) by 3.



while Loops

- Count the number of integers between 1 and 986 that are divisible (with no remainder) by 3.



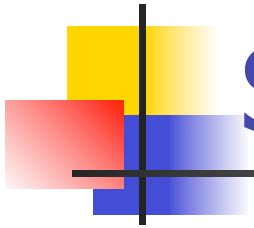


Types of M-Files

- There are two types of m-files:

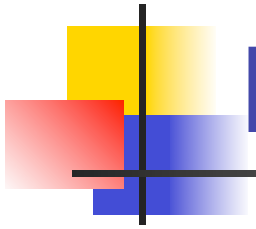
  Scripts and Functions

- Both are created in the Editor window
- Up to this point, we have been creating script files



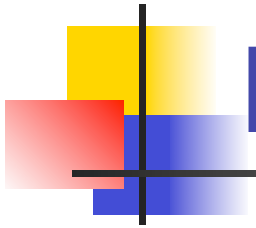
Script m-files

- Sequence of MATLAB commands
- Equivalent to typing a series of commands in the command window, except that scripts can be run at any time
- Cannot accept input
- All constants/values that need to be used should be defined at the beginning of the script file
- Values cannot be passed between script files.



Function M-Files

- Sub-program
- Can accept input and return outputs
- Creating a file that works just like a pre-defined MATLAB function (sin(x), mean(x)), where the input x is required
- Can extend the MATLAB language
- Can access functions from within other scripts and functions



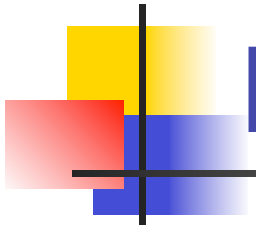
Function M-Files

```
function [x,y,...]=FunctionName(arg1,arg2,...)
```

- The name of the function (`FunctionName`) must be the same as the saved m-file name, and is used to call upon the function
- The input arguments for the function appear in parenthesis after the function name. You can have any number of arguments
- The m-file must begin with the function declaration
- If there are output arguments, they are specified in square brackets. If there is no output, leave the output blank.
- You can't run a function file from the Editor. You have to call the function in the command window:

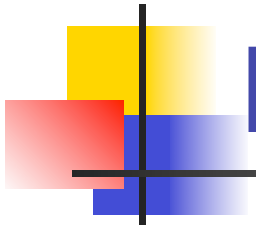
FunctionName(arg1, arg2)

Where arg1 and arg2 are the input values that will be used in the function



Function M-Files

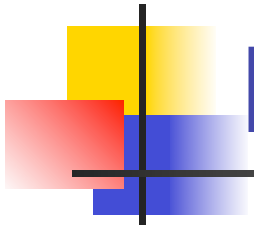
- Write a function that will calculate the hypotenuse of a right angle triangle, given the other two sides.



Function M-Files

- Write a function that will calculate the hypotenuse of a right angle triangle, given the other two sides.

```
function pythagoras(Side1, Side2)


Hypotenuse = sqrt((Side1^2) + (Side2^2))
```



Function M-Files

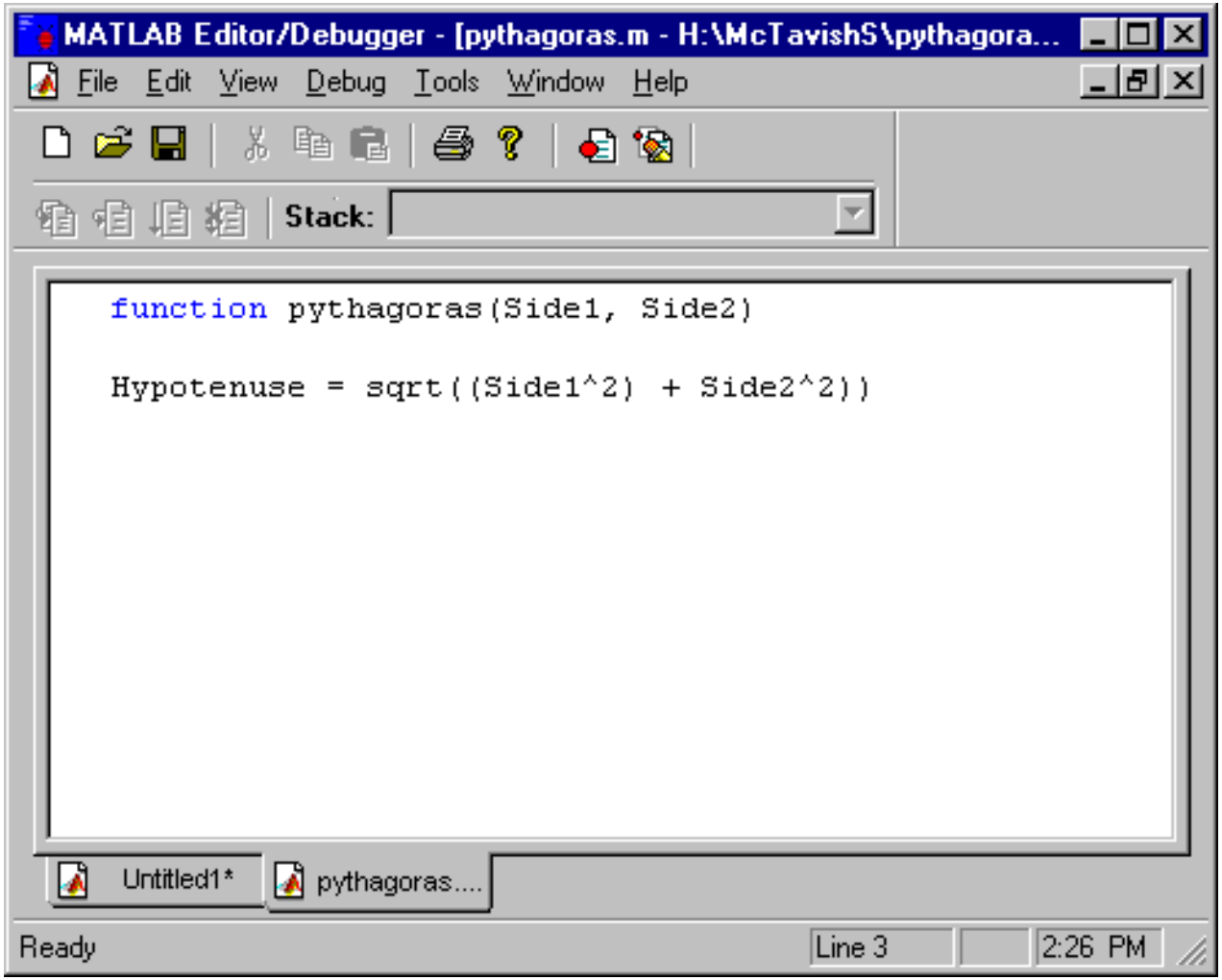
- Write a function that will calculate the hypotenuse of a right angle triangle, given the other two sides.

```
function pythagoras(Side1, Side2)


Hypotenuse = sqrt((Side1^2) + (Side2^2))
```

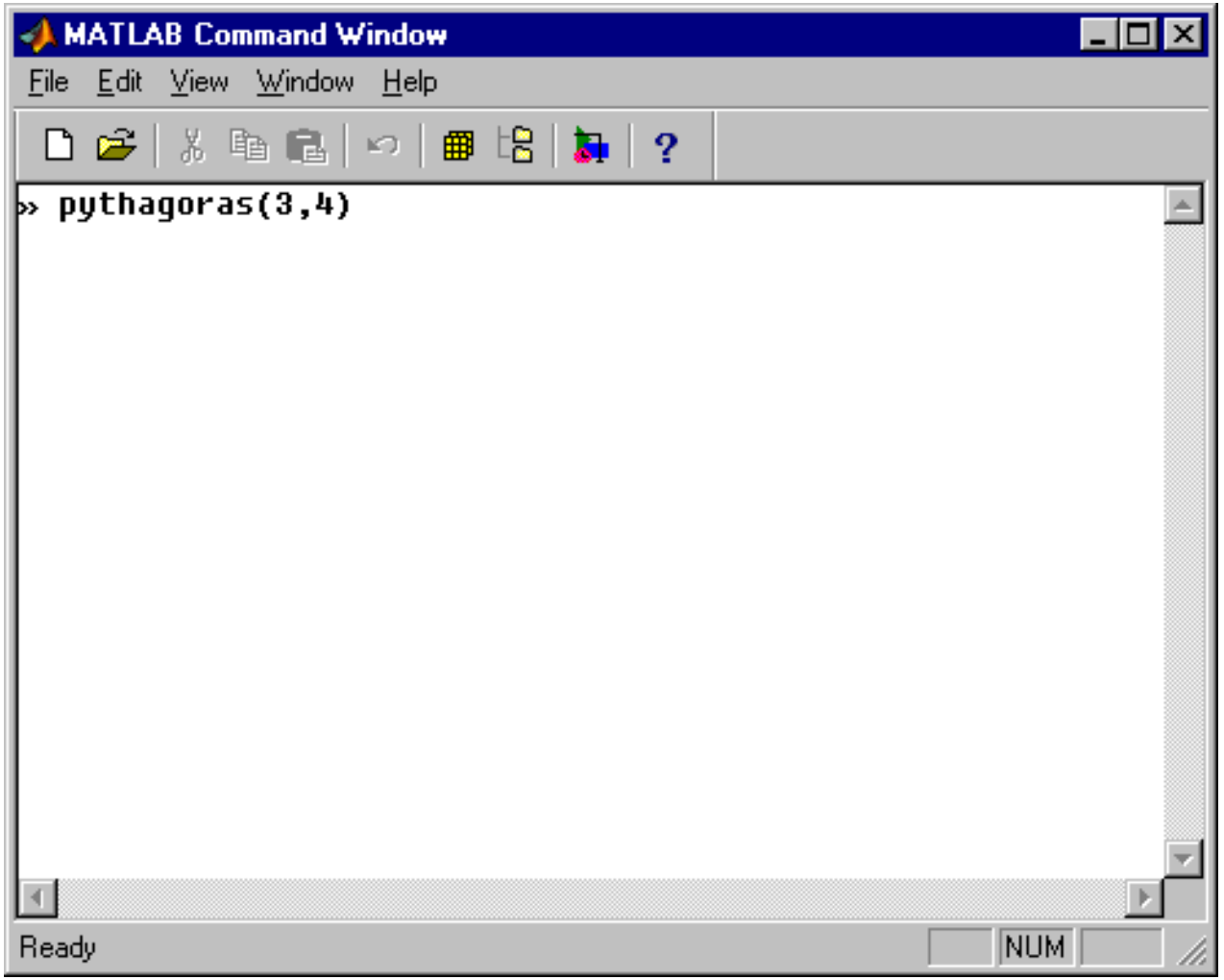
To use this function, simply type in the command window (or from within a script file):

pythagoras(Length of side1, Length of side2)

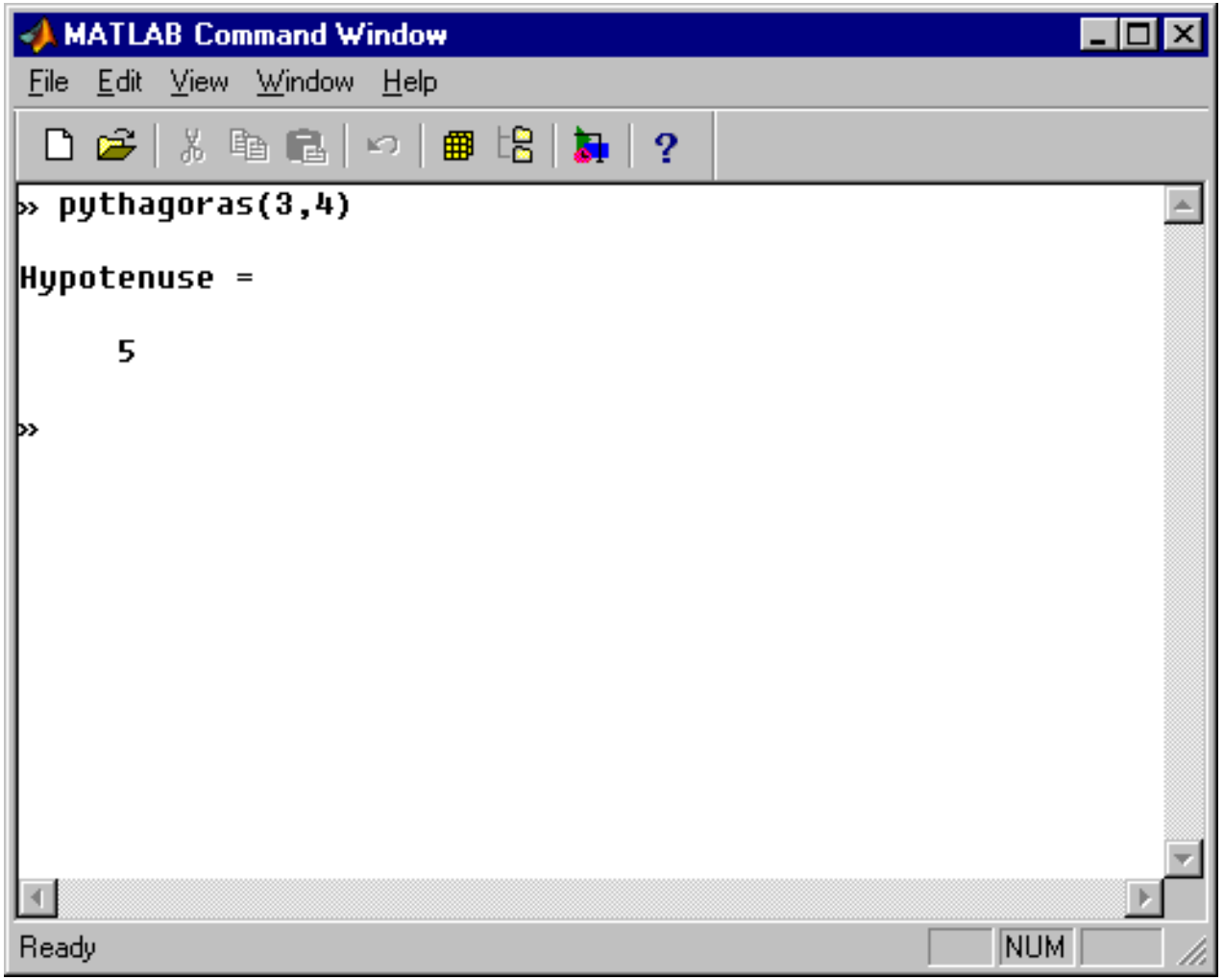
Functions


MATLAB Editor/Debugger - [pythagoras.m - H:\McTavishS\pythagora...
File Edit View Debug Tools Window Help
Stack:
function pythagoras(Side1, Side2)
Hypotenuse = sqrt((Side1^2) + Side2^2))
Untitled1*
pythagoras....
Ready
Line 3
2:26 PM

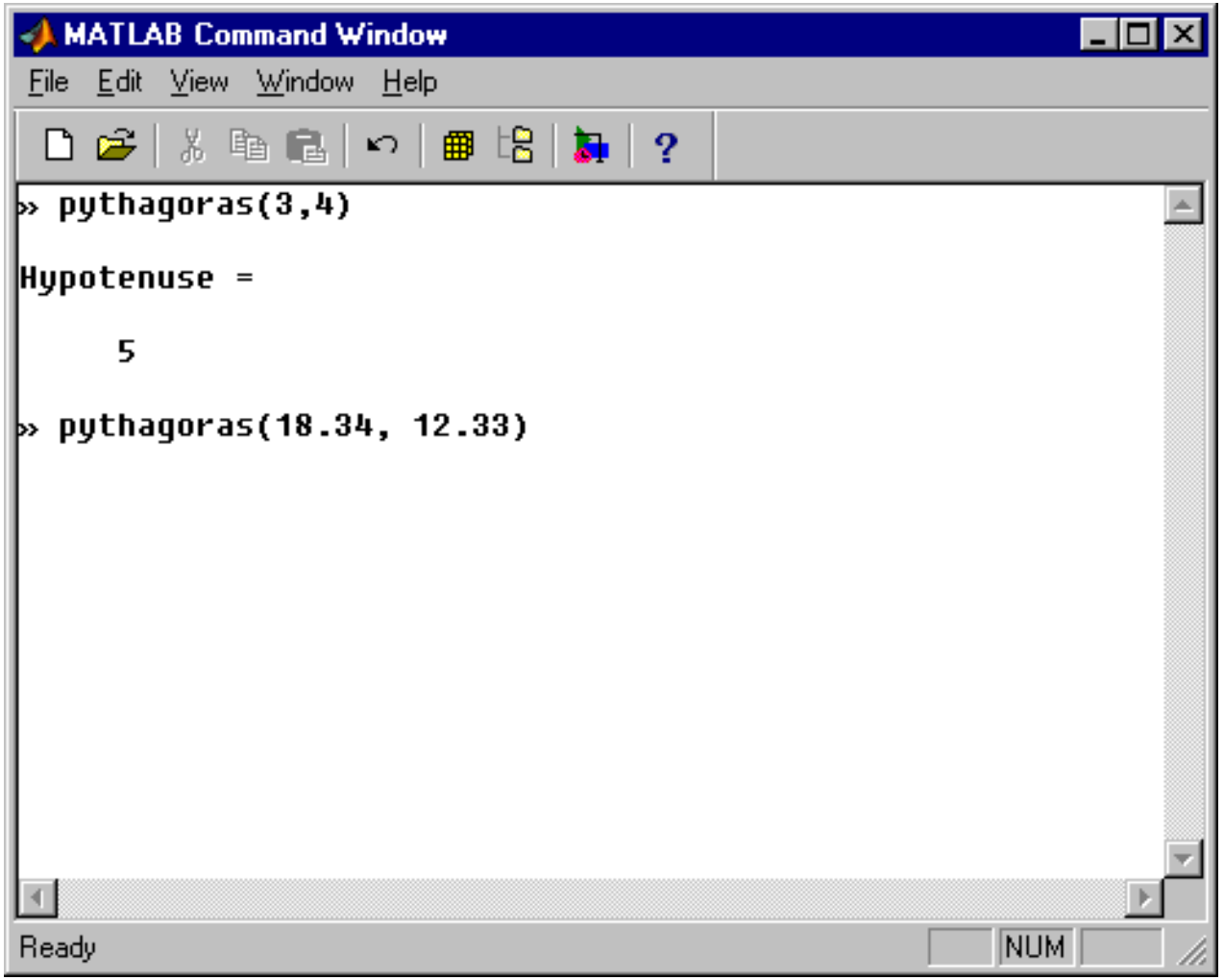
Functions


MATLAB Command Window
File Edit View Window Help
» pythagoras(3,4)
Ready
NUM

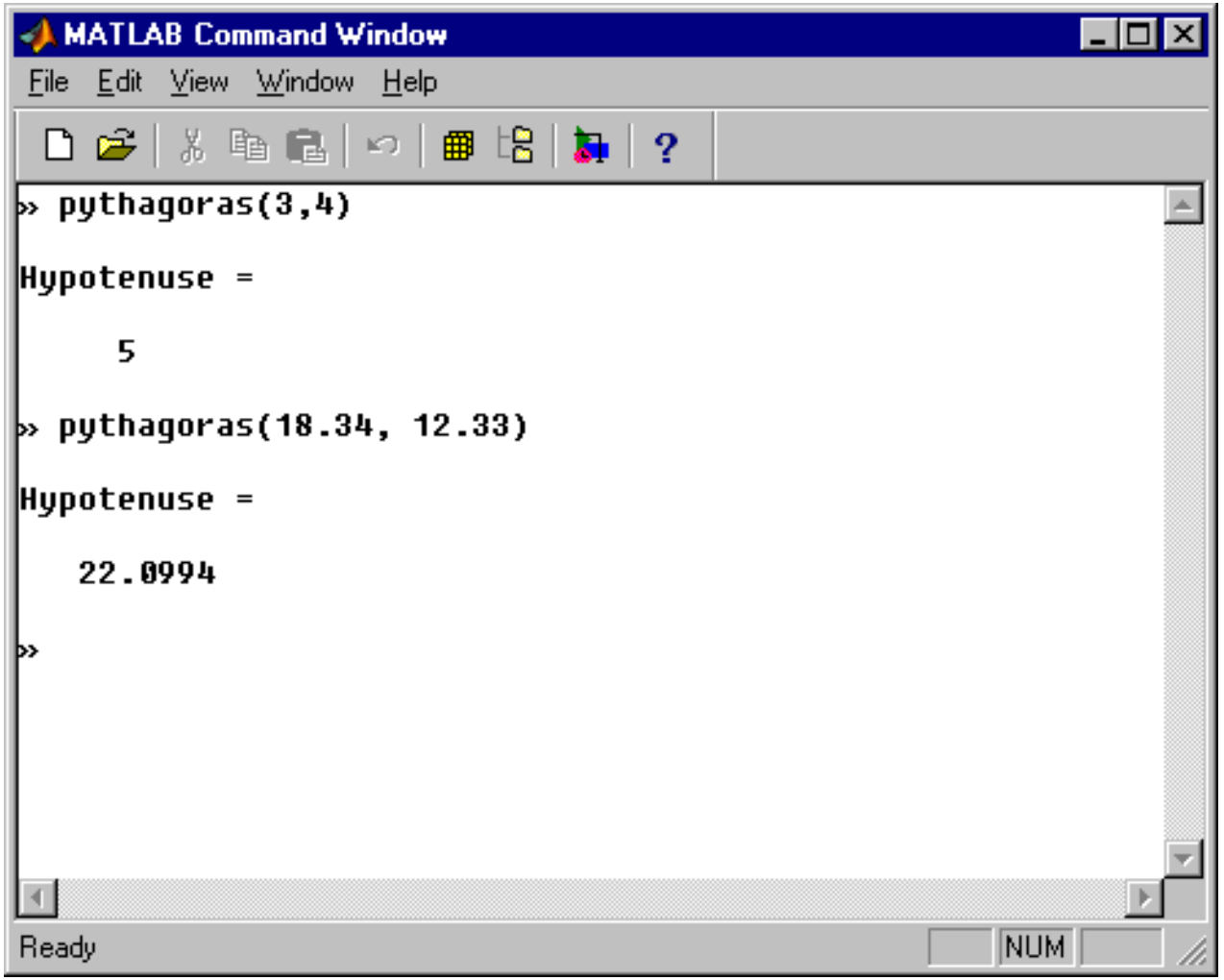
Functions

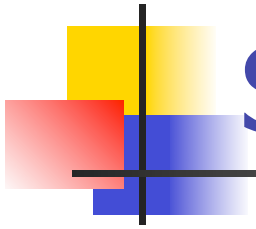

MATLAB Command Window
File Edit View Window Help
» pythagoras(3,4)
Hypotenuse =
5
»
Ready
NUM

Functions


MATLAB Command Window
File Edit View Window Help
» pythagoras(3,4)
Hypotenuse =
5
» pythagoras(18.34, 12.33)
Ready
NUM

Functions


MATLAB Command Window
File Edit View Window Help
» pythagoras(3,4)
Hypotenuse =
5
» pythagoras(18.34, 12.33)
Hypotenuse =
22.0994
»
Ready
NUM

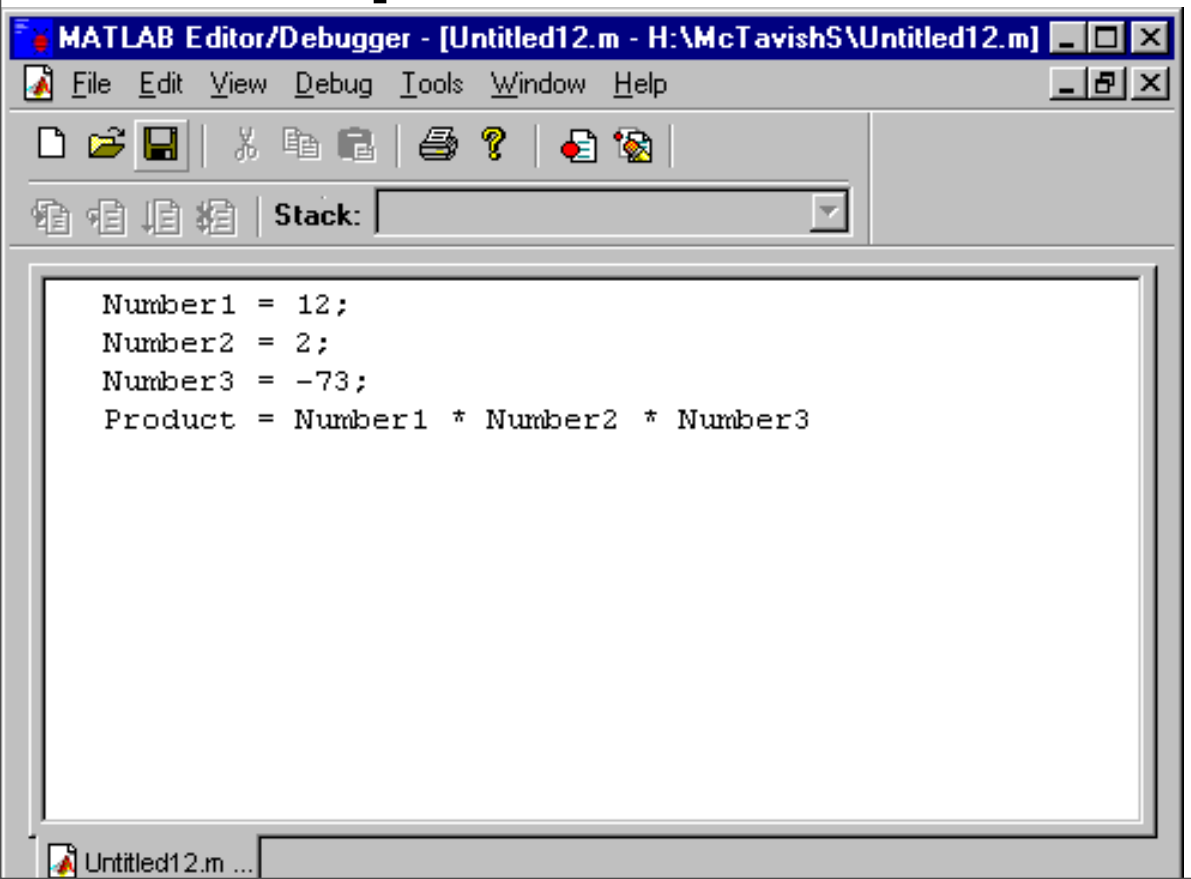


Script Vs. Function

- Write a script file and a function that will find the product of three numbers

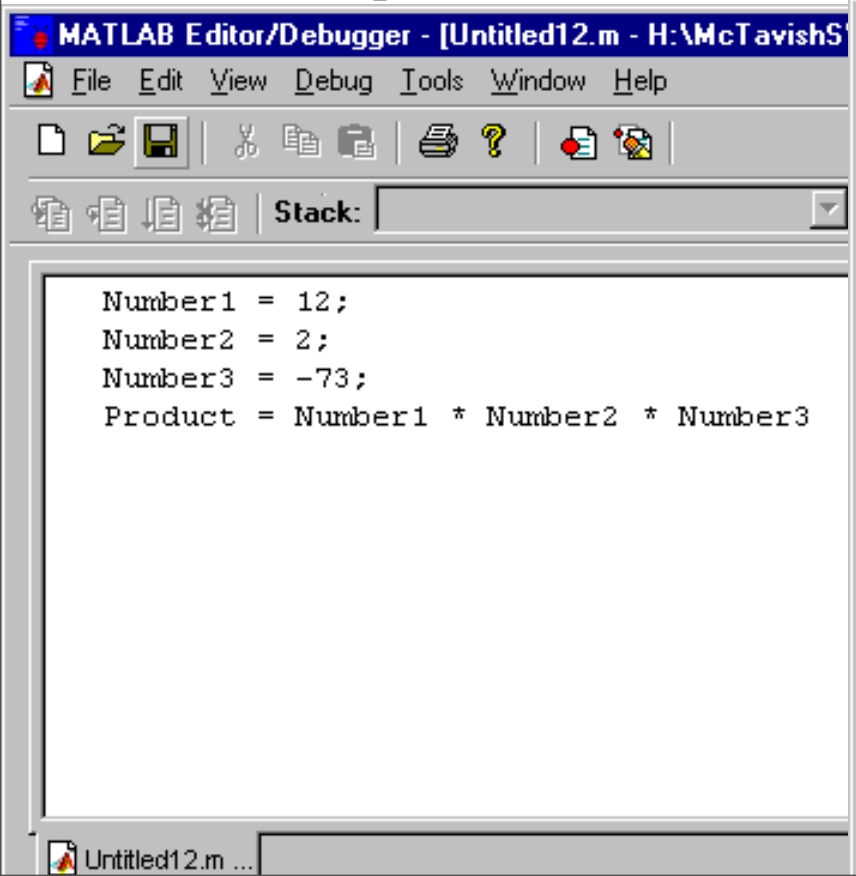
Script Vs. Function

Script m-file

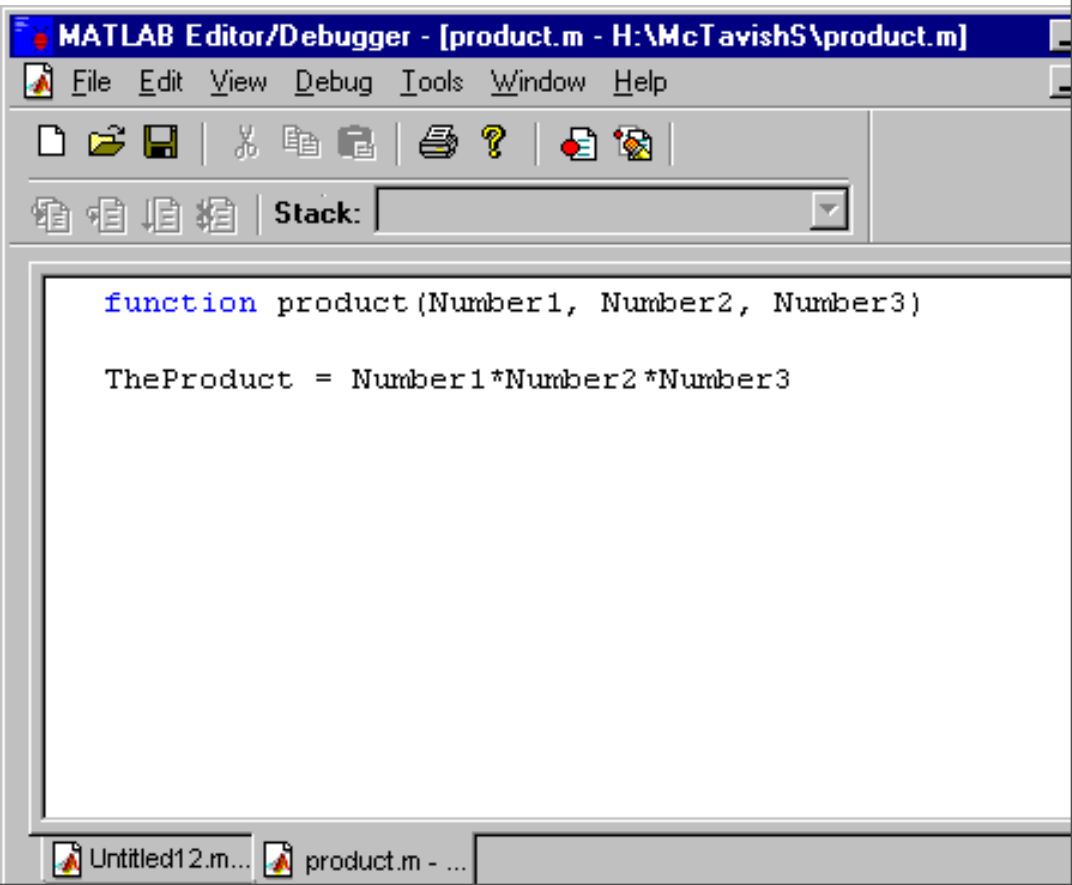


Script Vs. Function

Script m-file

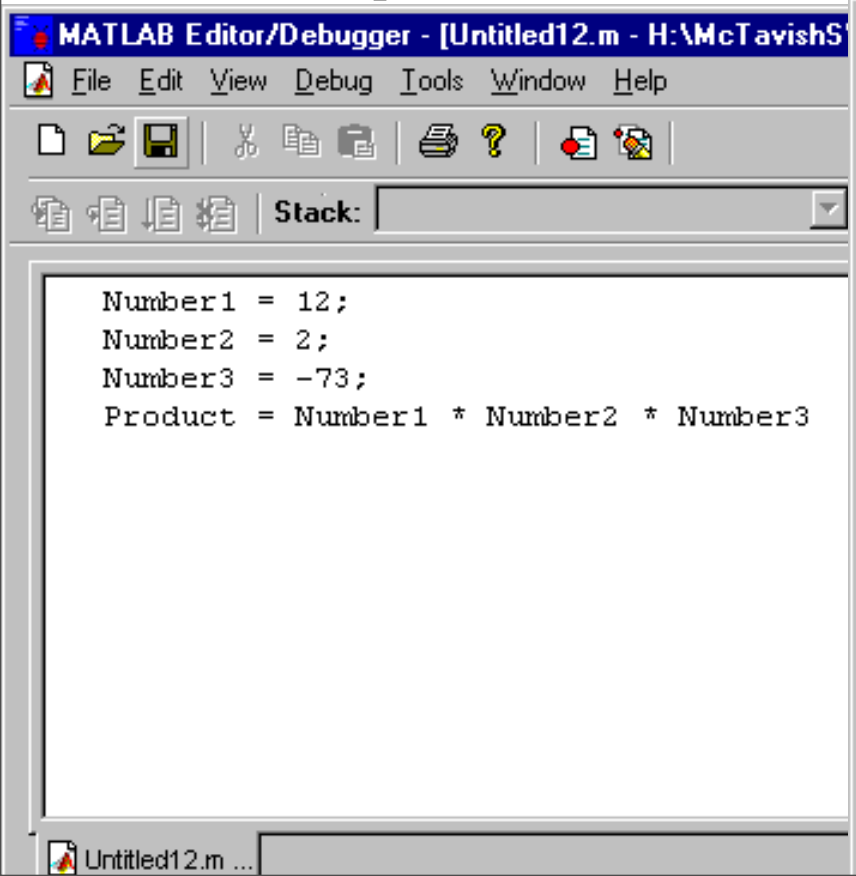


Function m-file

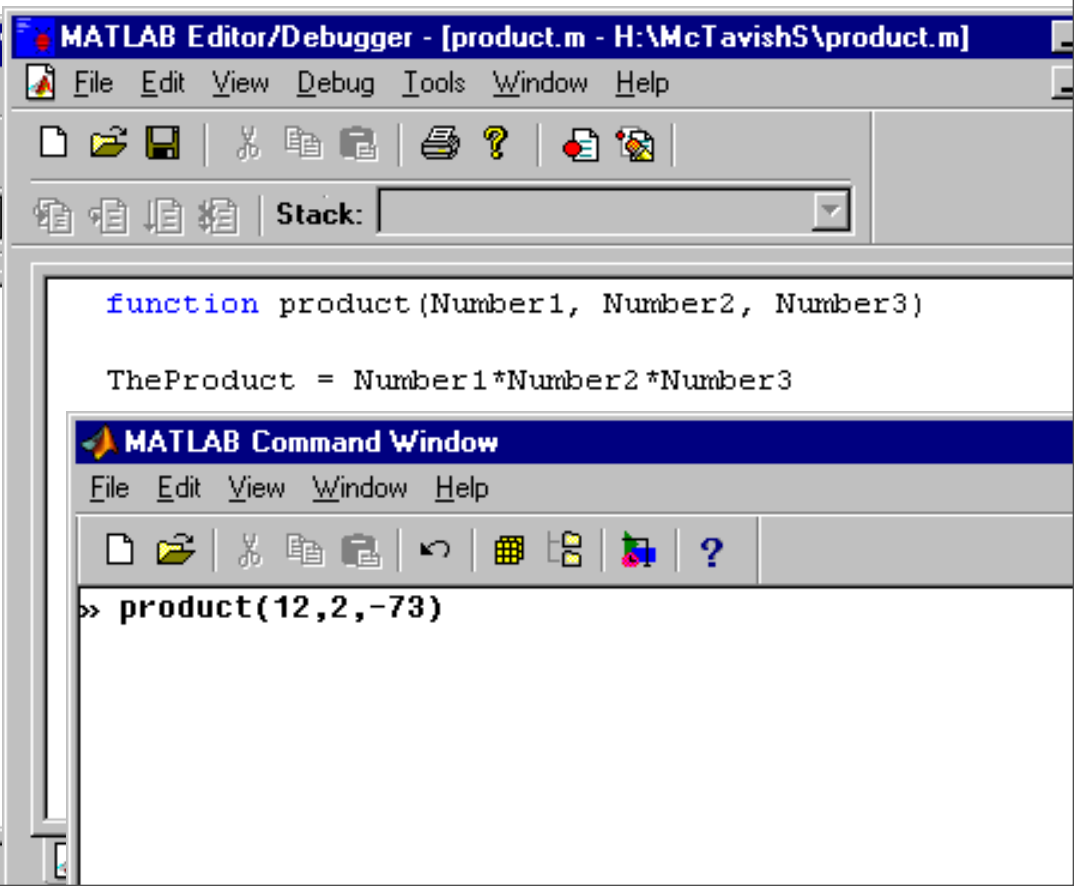


Script Vs. Function

Script m-file

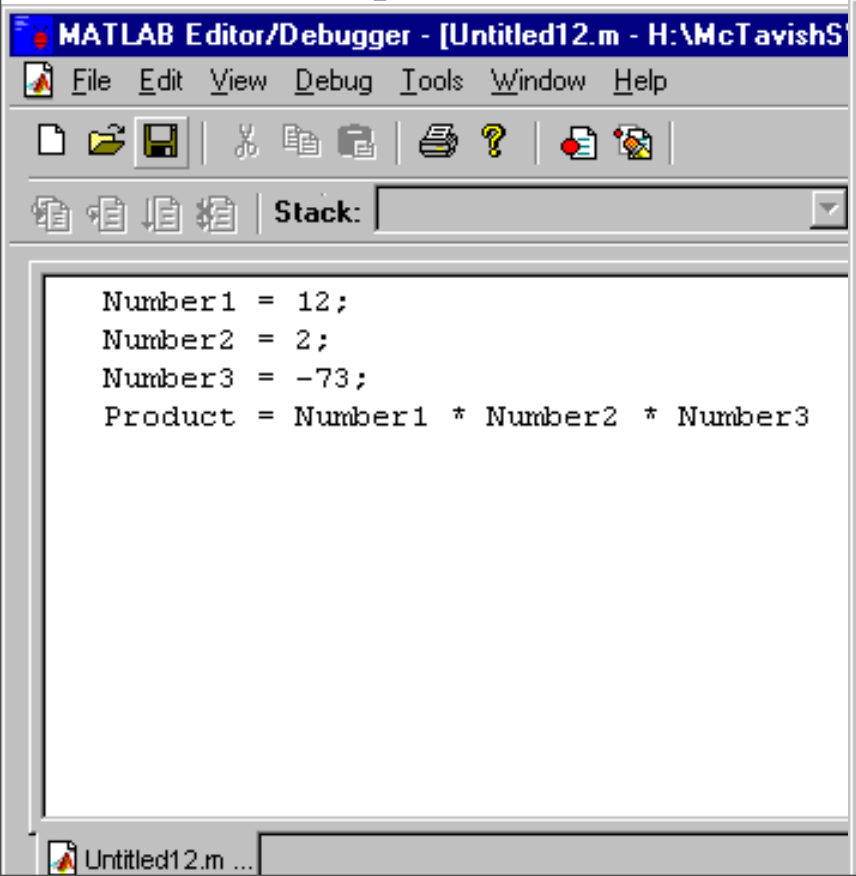


Function m-file

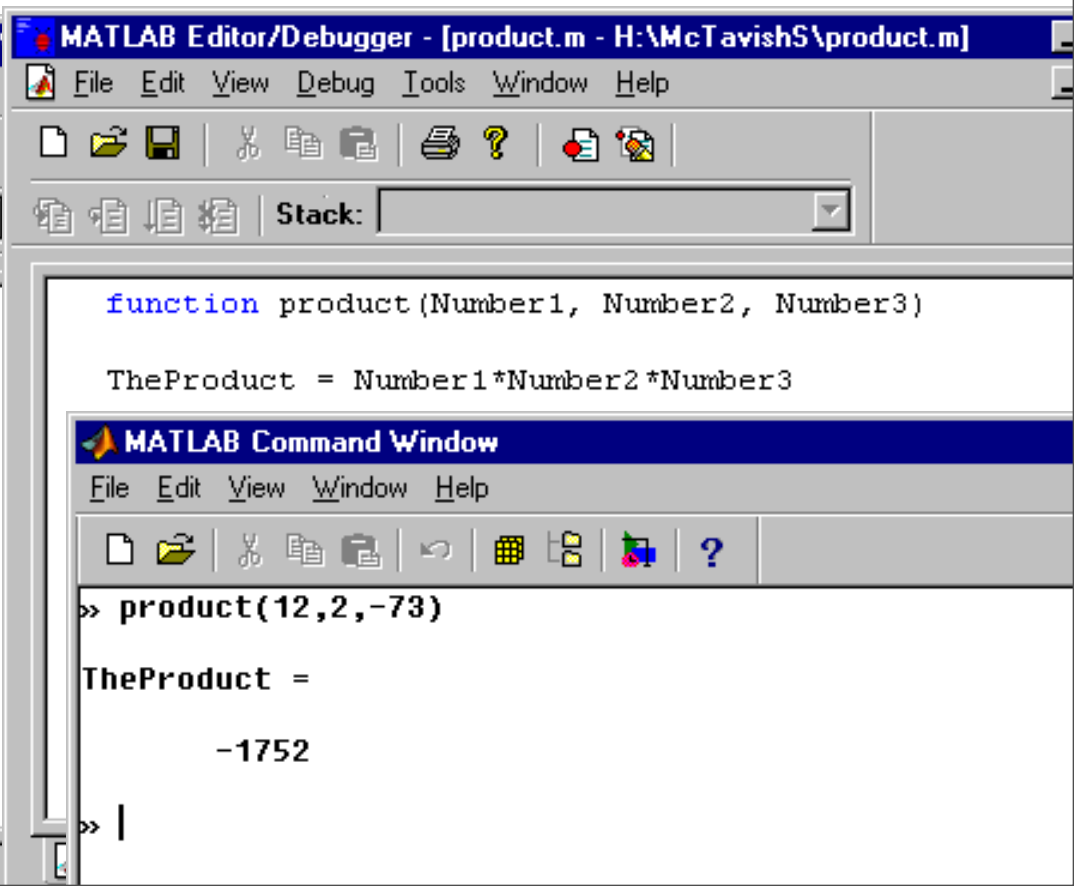


Script Vs. Function

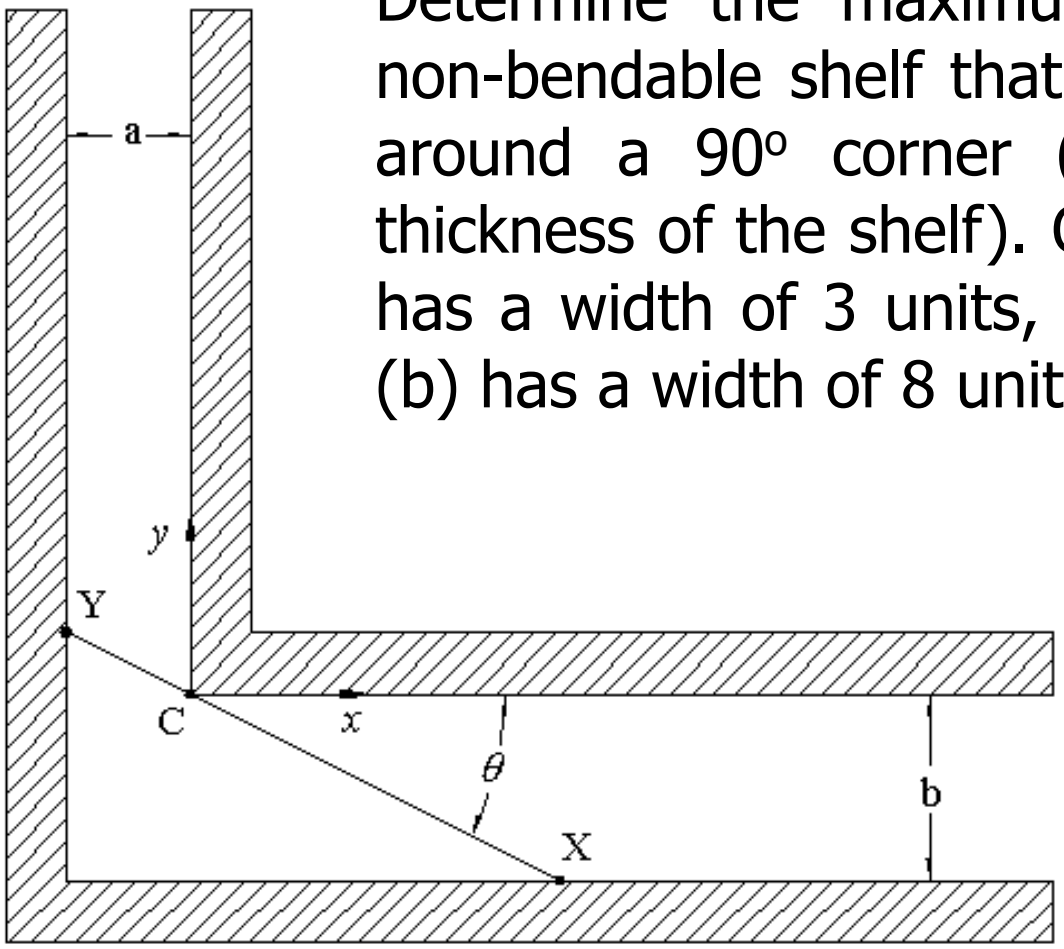
Script m-file



Function m-file



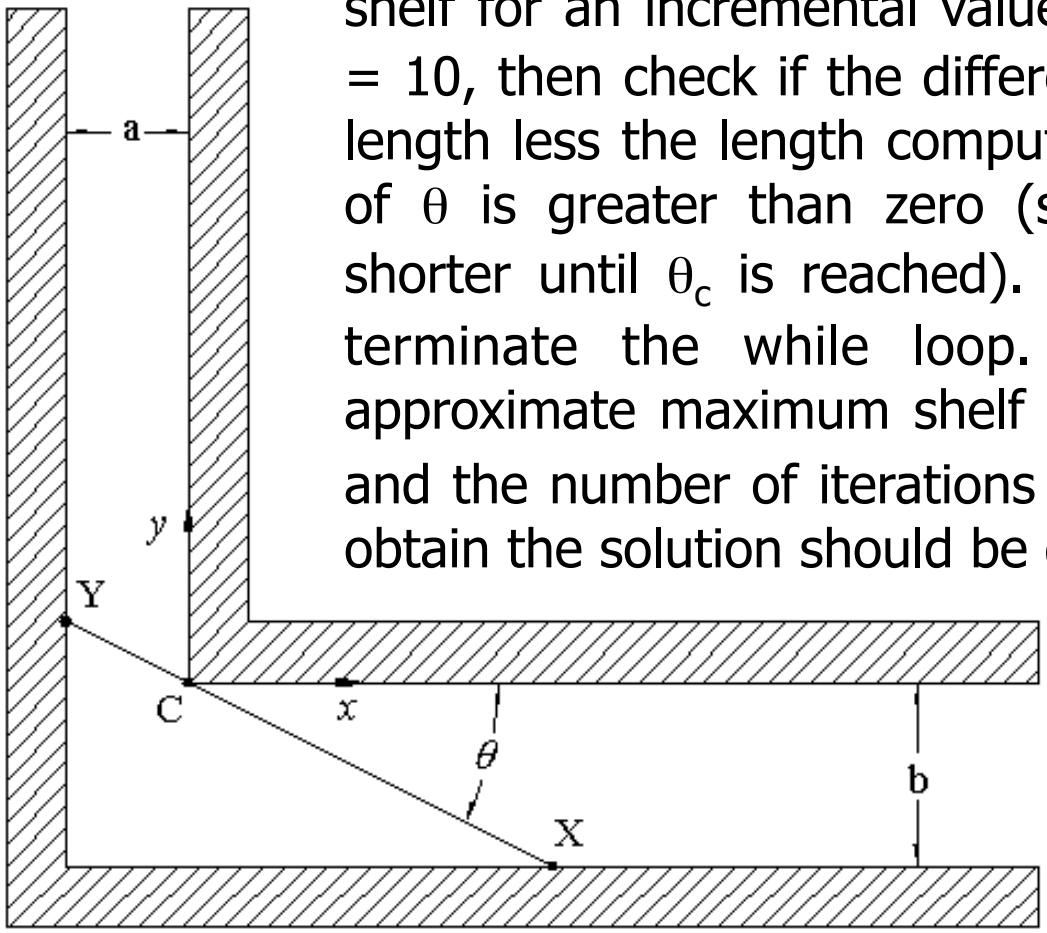
An Example



Determine the maximum length of a non-bendable shelf that can be carried around a 90° corner (neglecting the thickness of the shelf). One hallway (a) has a width of 3 units, while the other (b) has a width of 8 units.

An Example

Use a while loop. Calculate the length, YX, of the shelf for an incremental value of θ, starting with θ = 10, then check if the difference of the computed length less the length computed for the next value of θ is greater than zero (since the length gets shorter until θ_c is reached). This condition should terminate the while loop. The value for the approximate maximum shelf length, θ_c in degrees, and the number of iterations that were required to obtain the solution should be displayed.



MATLAB Editor/Debugger - [shelving.m - H:\McTavishS\shelving.m*]

File Edit View Debug Tools Window Help

Stack:

```
clear all;  % clears all previous variables
close all;  % closes all previously opened windows
clc         % clears the output screen
format long  % changes the number formatting for display

a = 3;      % sets the length a to 3
b = 8;      % sets the length b to 8

Angle = pi/18;  % sets the initial angle to 10 degrees in terms of radians

NewAngle = 0;   % Sets the value for the newly measured angle to 0
NewLength = 0;  % Sets the length of the New Line to 0

Delta = 1; % Sets the difference of the length - new length to 1

Iterations = 0; % Sets the counter for the number of iterations to 0
Step = 0.0001;  %This is amount by which each angle will increase (in radians)

Length = a/cos(Angle) + b/sin(Angle);   %Determines the length of the material

while (Delta > 0)  % begins the while loop and sets the end condition

    NewAngle = Angle + Step; % Adds the step value to the angle

    NewLength = a/cos(NewAngle) + b/sin(NewAngle); % calculates the
                                                   %lenght of the new line
```

pythagoras.... shelving.m - ...

Ready Line 12 10:47 AM

MATLAB Editor/Debugger - [shelving.m - H:\McTavishS\shelving.m*]

File Edit View Debug Tools Window Help

Stack:

```
    NewLength = a/cos(NewAngle) + b/sin(NewAngle); % calculates the
                                                   %lenght of the new line

    Delta = Length - NewLength; % evaluates the new value of Delta

    Length = NewLength; % Sets the value of length to newlength
    Angle = NewAngle; % sets the value of angle to newangle

    Iterations = Iterations + 1; % increases the iteration counter by 1

end


Iterations %displays the final number of iterations

CriticalAngle = NewAngle %displays the critical angle in radians

AngleInDegrees = (180*CriticalAngle)/pi %displays the critical angle in degrees

NewLength % displays the ideal length of the material that will fit
          %around the corner
```

pythagoras.... shelving.m - ...

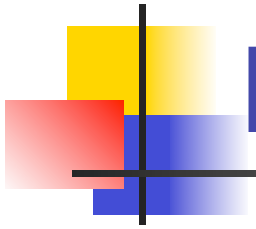
Ready Line 12 10:47 AM

MATLAB Command Window

File Edit View Window Help

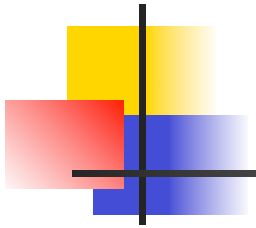
```
Iterations =

        7716


CriticalAngle =

   0.94613292519935


AngleInDegrees =

  54.20942347228946


NewLength =

  14.99216511329710
```

Ready NUM



Help?

- For extra help, see the MATLAB support files on WebCT



Reading Assignment

Chapters 18 and 19