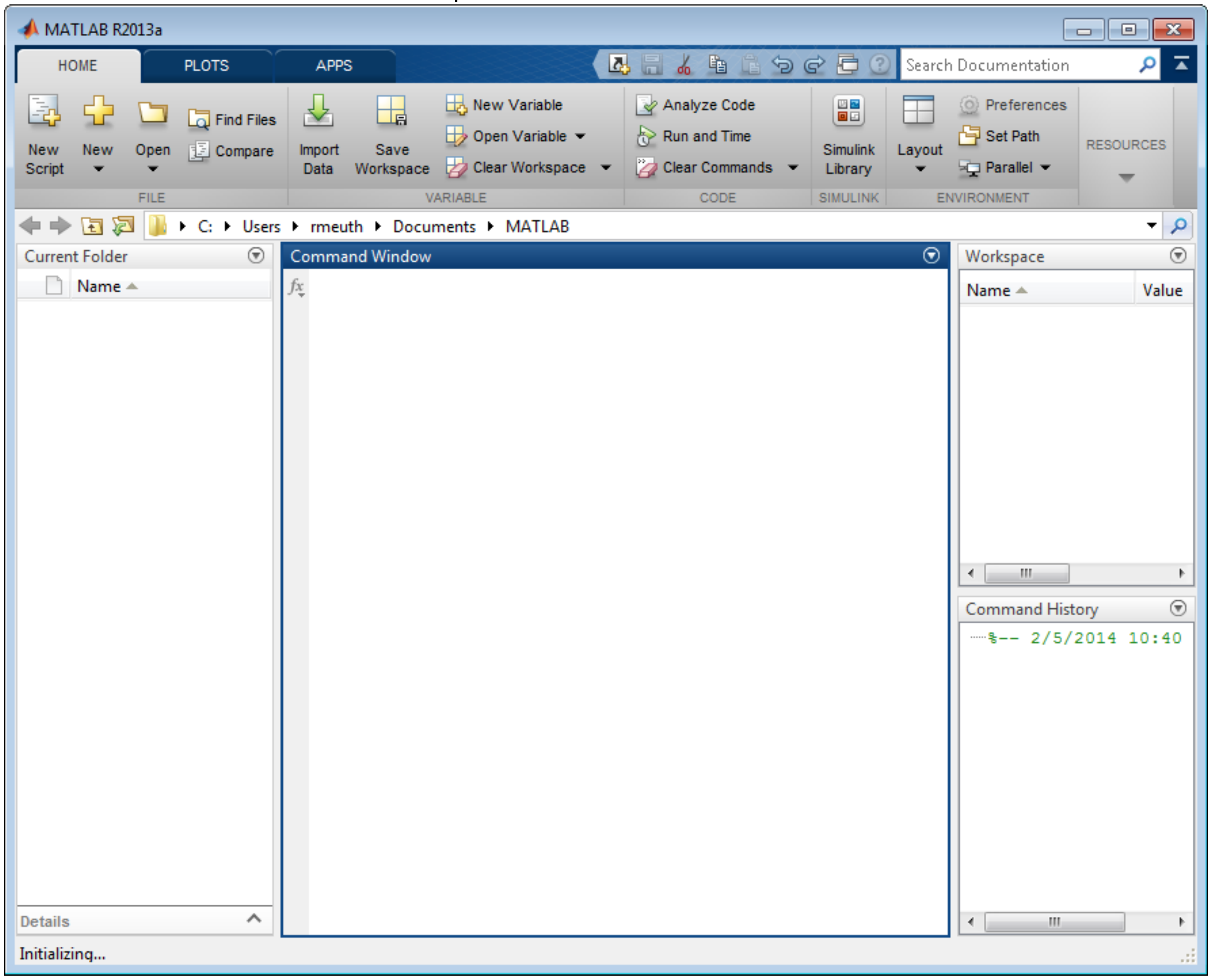
Skills Module: Introduction to Programming with MATLAB

Start MATLAB by opening the Start Menu-> All Programs, then MATLAB R2016a.

A window similar to this should show up:



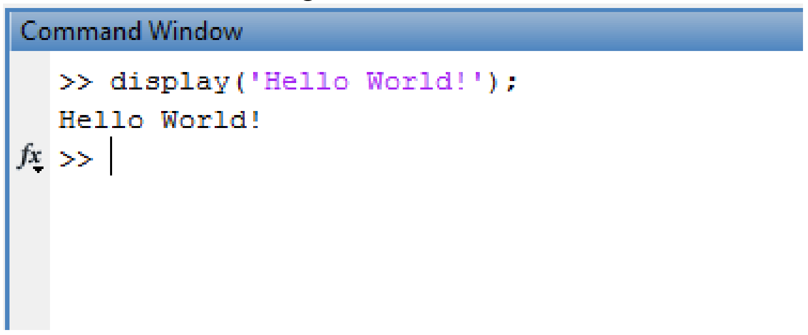
The *Command Window* is the primary interface to MATLAB. It is where you can quickly execute commands and see the results of calculations and the output from programs.

Hello World

In the *Command Window*, type the following, then hit the Enter key:

```
display('Hello World!');
```

You should see something like this:



```
>> display('Hello World!');
Hello World!
>> 
```

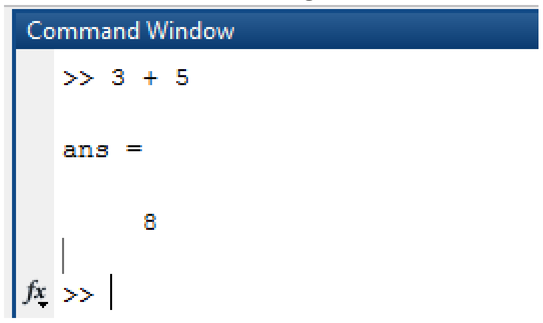
The display function can be used to display text and numeric expressions.
The disp function is a simpler alternative when you just want to display the value of a variable.

MATLAB as a Simple Calculator

We can use MATLAB as a calculator. In the *Command Window*, type the following and hit Enter:

```
3 + 5
```

You should see something like this:

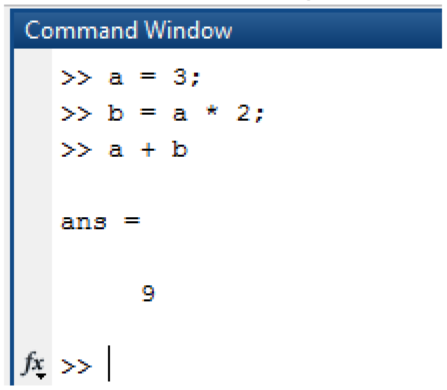


```
>> 3 + 5

ans =

     8

>> 
```

We can also store values in *variables* and then use those variables in calculations. To assign a value to a variable we use the syntax: `variable_name = value` (the value can be *expression* like **a * 2** below). Try the following:

```
a = 3;
b = a * 2;
a + b
```

You should see something like this:

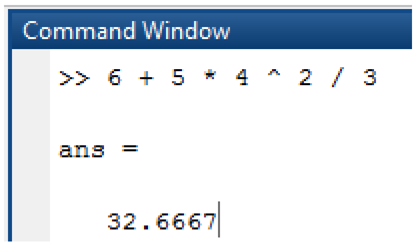


```
>> a = 3;
>> b = a * 2;
>> a + b

ans =

     9

>> 
```

Expressions always evaluate to a value. The expression **a * 2** in the above example evaluates to **6** because **a** is assigned the value **3** in the first line, and **3 * 2** is **6**, so the value **6** is assigned to the variable **b**.

Also, notice that when we append a semicolon (**;**) at the end of a command, the result is not displayed to the *Command Window*, but when we omit the semicolon, MATLAB displays the result.

We can also perform more complex calculations, like $6+5\times\frac{4^2}{3}$
But we need to render such expressions in a linear form like this: **6 + 5 * 4 ^ 2 /3**



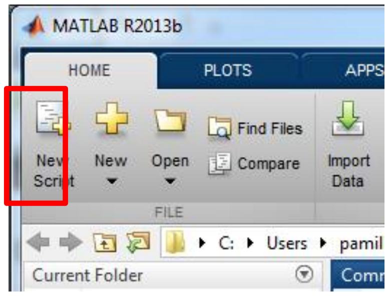
```
>> 6 + 5 * 4 ^ 2 / 3

ans =

   32.6667
```

Notice that MATLAB follows the standard order of operations. You can also use parentheses.
Also Notice that MATLAB uses the ^ symbol (call a caret) for exponentiation.

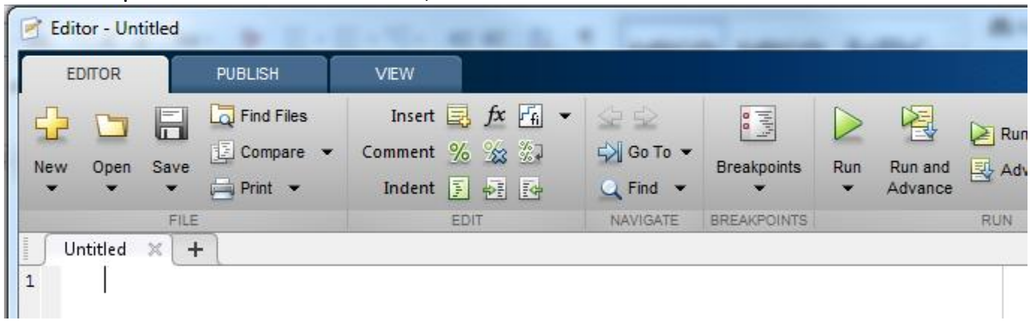
Scripts or M-Files

Using the *Command Window*, we can enter commands to execute, one at a time, which is useful for experimenting and prototyping commands to seeing their results. But often we want to execute several or many commands (a program) easily, or to be able to re-execute a bunch of commands without having to retype or copy and paste them. To do this we need to create a new MATLAB script or M-file. MATLAB script files have a `.m` file extension.

To create a blank script file, go to *File->New->Script* or click the *New Script* Button to create a new M-file.



This will open a new *Editor* window, and a blank unsaved document.



This is where we can write a sequence of commands to be executed in order. To test this, copy the following into the document:

```
a = 3;
b = a * 2;
c = a + b;
disp(c)
```

Save and name the file **`MyFirstScript`**, then click the *run icon*, or press F5 to execute the script. Note that the `disp` function tells MATLAB to display a value in the Command Window. Also, you should know that, if you are using one of the ASU Lab computers, any scripts that you write may be deleted when you log off the machine - so **you should always save a copy of any important script files to an external source** like a flash drive or your ASU Google Drive account.

Sometimes we want to get input from the user through the keyboard. We may also want to produce more complex outputs that are formatted to be easier for the user to read. MATLAB has many built in commands (or *functions*) for such purposes.

Read about the **`input`** function here: http://www.mathworks.com/help/matlab/ref/input.html
Read about the **`sprintf`** function here: http://www.mathworks.com/help/matlab/ref/sprintf.html

Test Your New Skills

Write a program that helps a user calculate the amount to tip a waitperson at a nice dinner. The program should ask the user for the check amount, then display the amount to tip for 10%, 15% and 20%, including charge totals for each. An example execution would look like this:

```
Enter the amount of the check: $ 45.16

For 10%, tip $4.52 for a total of $49.68
For 15%, tip $6.77 for a total of $51.93
For 20%, tip $9.03 for a total of $54.19
```

Variables and While Loops

Variables

In the section titled MATLAB as a Simple Calculator, you created and used variables named **a**, **b**, and **c**:

```
a = 3;
b = a * 2;
c = a + b;
disp(c)
```

Note that you do not have to *declare* the variable or its *type* (int, char, float, etc.) as you may be used to from programming languages like Java or C++,. In MATLAB, variables are created when we first assign a value to them, and the type of a variable is determined by the data that we assign to it.

For example:

```
x = 1;                  % Integer Type
y = 'Jolly Rancher';    % String Type
z = 5.78;               % Floating point type.
```

Note that the percent sign (**%**) denotes a comment in MATLAB code. Comments in the code are ignored by the computer; they are there to annotate and describe some aspect of the code for human readability. Another way to help make your code more readable is to use descriptive variable names - instead of variable names like **a**, **b**, and **c** we should use names like **`check`**, **`tip`**, and **`total`**.

We can also re-assign new values to variables. In the following example, we assign the value **3** to the variable **a**, then we assign the value **6** to the variable **b**. Then we assign a new value **9** to the variable **a**.

```
a = 3;
b = a * 2;
a = a + b;
display(a)
```

A variable is called a "variable" because we can assign new and different values to it whenever we want - so, the value varies.

While Loops

Sometimes we want to execute a set of instructions more than once. We could simply write the instructions multiple times. For example, suppose that we want to calculate the total of three numbers that the user will input in the console. We could do something like this:

```
total = 0;
number = input('Enter a number: ');
total = total + number;
number = input('Enter a number: ');
total = total + number;
number = input('Enter a number: ');
total = total + number;
display(total)
```

And if we wanted to calculate the total of ten numbers then we could repeat the instructions to collect the input and add to the **total** ten times. But this becomes tedious and error prone. A better solution is to use *loops* to execute a set of instructions more than once.

Start a new script, then write and execute the following:

```
i = 0;
while i < 5
    disp(i);
    i = i + 1;
end
```

How many times is the number displayed?

Change the **while** line to the following and then re-run the script:

```
while i < 8
```

How did that change the output?

Next, change the increment line to:

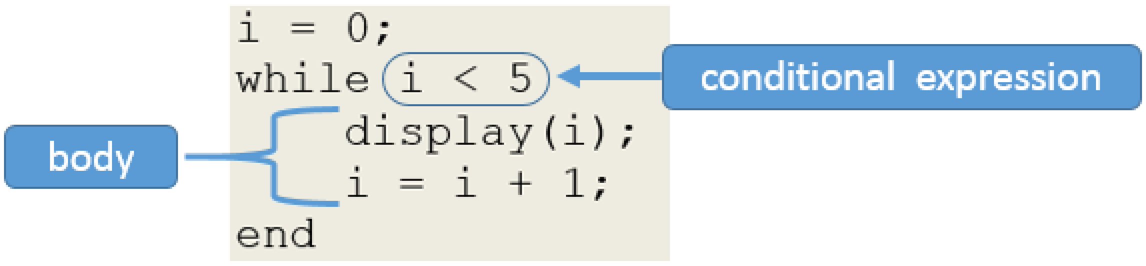
```
 i = i + 2;
```

How did that change the output?

Note that every **while** statement requires a corresponding **end** statement. The instructions between the **while** and the **end** are called the *body* of the loop. The body is the set of instructions that may be executed more than once.

In the MATLAB Command Window type **help while** and press Enter. This is an alternative way to find information about MATLAB functions.

The code following the **while** keyword is called a *conditional expression*, and such expressions always evaluate to either **true** or **false**. If the conditional expression evaluates to **true**, then the body of the loop (the code between the **while** keyword and its corresponding **end** keyword) will be executed. After the body of the loop is executed the conditional expression will be tested again. If the conditional expression is **false**, then the body of the loop is skipped, and the code that follows after the **while** loop's **end** statement starts executing.



```
i = 0;
while (i < 5)
    display(i);
    i = i + 1;
end
```

Try modifying the program above and observe the results.
HINT: Pressing **Ctrl + C** at the same time kills a program that is running forever. This is not a good way to stop a program, but sometimes it is necessary.

Conditional Expressions

Conditional Expressions often involve comparing two values - like **i** and **5** in the while loop conditional expression above. There are several different comparisons that you can use. There is a different *relational operator* for each type of comparison.

The Relational Operators (below) compare two values (the values can be expressions) and the result is either **true** or **false**.

Operator	Description
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
~=	Not equal to

The relational operators can be use with *Logical Operators* to combine **true** and **false** values. Note that in MATLAB **true** is represented by the value **1** and **false** is represented by the value **0**.

Inputs A and B		and A && B	or A || B	xor xor(A,B)	not ~A
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

For example, consider the following code:

```
i = 0;
z = 10;
while i < 3 && z >= 5
    i = i + 1;
    z = z - 2;
    disp(i + z)
end
```

The code inside the while-loop will only execute if *both* `i < 3` and `z >= 5` are `true`. If one of them is `false`, then the entire expression is `false` and the loop body is skipped.

Test Your New Skills

In a new script called **`MinEvens.m`**, write a program that takes two positive numbers as input, A and B, then displays the even numbers counting up to the smaller of the two numbers.

An example output of the program should look like this:

```
Enter number A: 7
Enter number B: 10
0
2
4
6
>>
```

If Statements and For Loops

If Statements

Start a new script, then write and execute the following:

```
number = randi(20); % pick a random number between 1 and 20

if number == 12
    disp('It is a dozen')
end

if number > 12 && number < 20
    disp('It is a teen')
end

if number >= 10 && number <= 12
    disp('It is between 10 and 12')
end

if number < 10
     disp('It has only one digit')
else
     disp('It has two digits')
end

display(number)
```

Run the script a few times to understand how it works. We can use **`if`** statements to get the computer to make a decision. Note that no parentheses are necessary around the conditional expression. The **`else`** part of the **`if`** statement is optional, but the **`end`** is always required. The code following the **`if`** is a *conditional expression*, and it follows the same rules as conditional expressions for **`while`** loops, but in this case, the body of the **`if`** statement is executed only once. If we want it to repeat, we have to use a loop like the **`while`** loop.

Note that the **`randi`** function generates a random integer between **`1`** and the argument value (**`20`** in this case).

Test Your New Skills

In a new script called **`GuessingGame`**, Write a program that does the following when run:

1. Pick a random secret integer between 1 and 100
2. Continue to prompt the user to input a guess until they guess the secret integer
3. After each user guess display a message to the user teeing then if their guess is:
    a. Too low
    b. Too high
    c. Correct
4. After the user has guessed correctly, display the number of tries the user took to guess correctly.

For Loops

Start a new script file in MATLAB and copy and execute the following code:

```
for i = 1:8
    disp(num2str(i));
    disp('nana');
end
disp('Batmaaaan!');
```

This is how we write **`for`** loops in MATLAB. In the command window, count the occurrences of **`'nana'`**. Note that the **`for`** loop executes through the range **`1`** to **`8`**, inclusively - in other words, the body of the loop executes 8 times, and the value of the variable **`i`** changes with each execution.

Note the syntax of the **`for`** loop. Every **`for`**, **`if`**, and **`while`** statement should have a corresponding **end** statement. Because of this, newlines (returns) and tabs are extremely important to make the code readable. Follow the pattern above in all cases.

HOT TIP: If your code is messy, press **`Ctrl + A`**, followed by **`Ctrl + I`** to automatically format your code.

Next, modify your script to look like this:

```
for i = 1:8
    if mod(i, 2) == 0 % If remainder of i/2 is equal to 0
        disp('ding');
    else
        disp('dong');
    end
end
display('The witch is dead!');
```

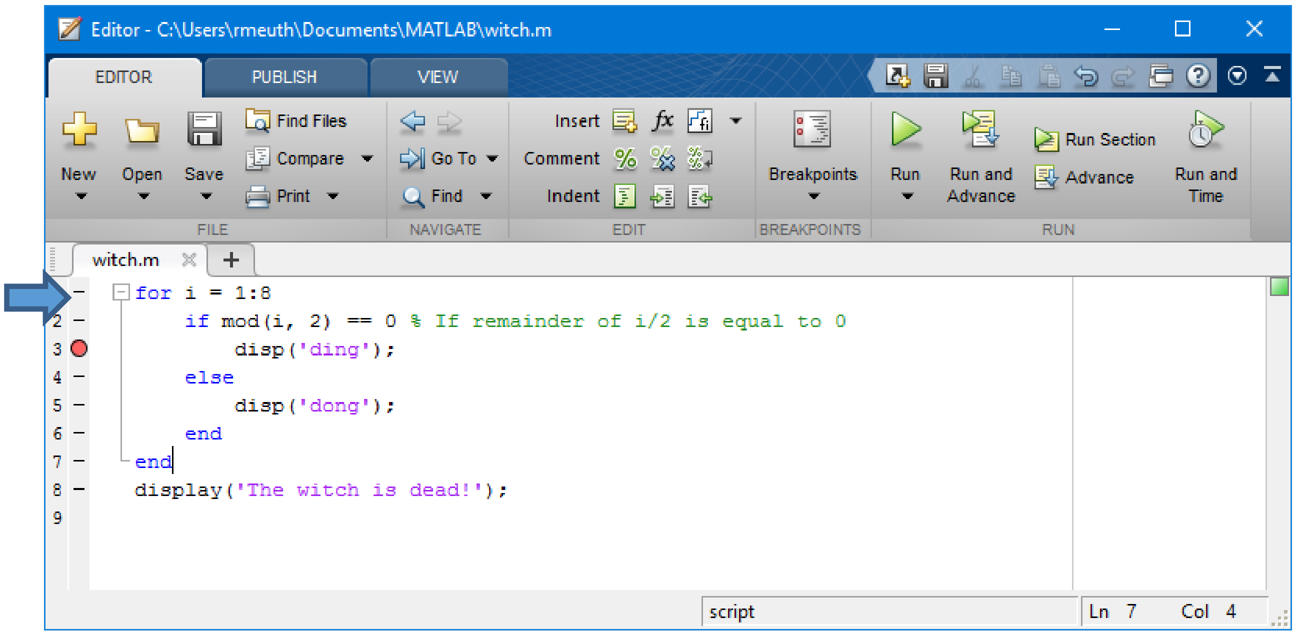
Note that we can nest an **`if`** inside of a **`for`** loop. In fact we can nest any **`if`** or **`for`** or **`while`** inside of any other **`if`**, **`for`**, or **`while`**.

Here we are also using a modulus function (**mod**). This function takes two parameters (**`i`** and **`2`** in this case), and calculates the remainder of the first number divided by the second. Note that parentheses are always required when using functions.

When you run this code, why does it output `'dong'` first instead of `'ding'`?

It might help to use the ***debugger*** to find out.

The MATLAB Debugger


Editor - C:\Users\rmeuth\Documents\MATLAB\witch.m

```
for i = 1:8
    if mod(i, 2) == 0 % If remainder of i/2 is equal to 0
        disp('ding');
    else
        disp('dong');
    end
end
display('The witch is dead!');
```

Click on the dash next to the number of the code like that contains “`disp('ding');`”.

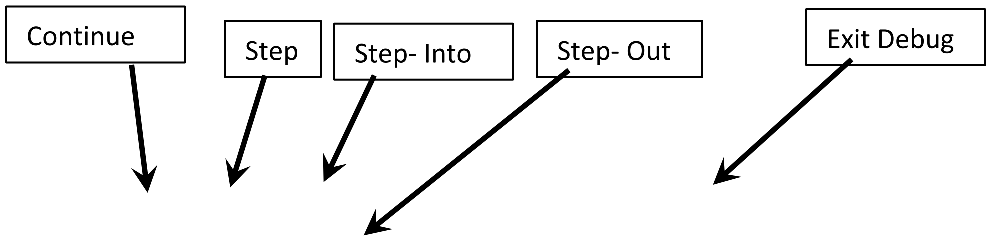
The dash should turn into a red dot. If the dot is gray, save the document first.
This sets a **Breakpoint**, which means that when execution passes to this line, the program will pause, and you will be able to examine the state of the program at that moment.

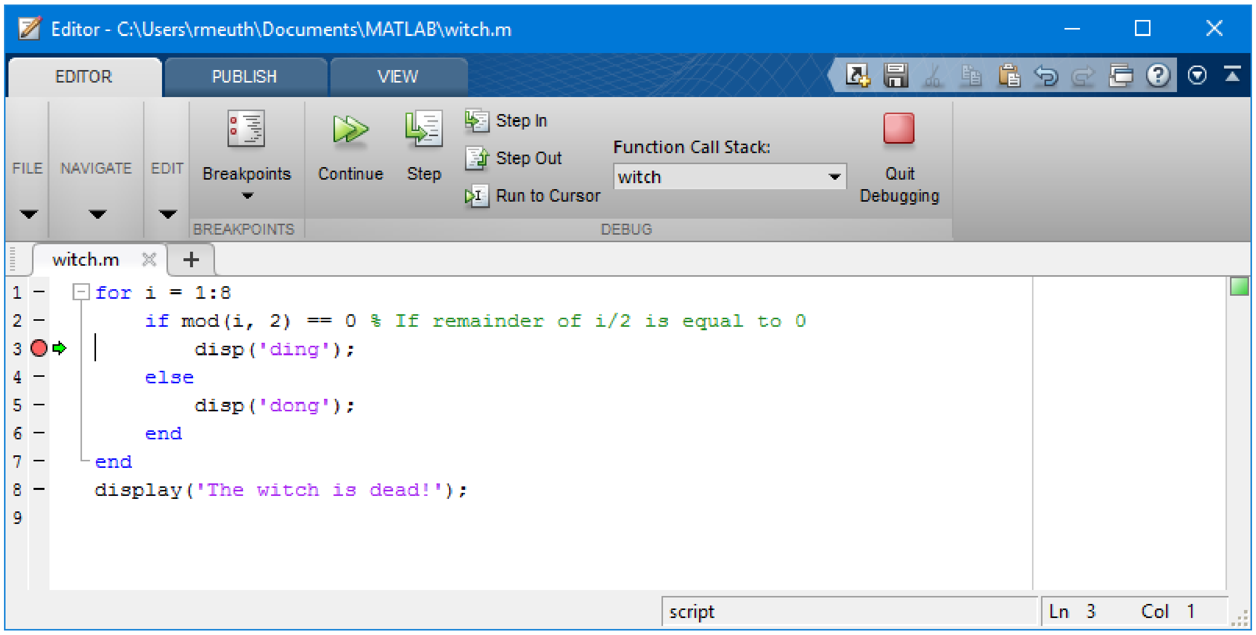
Do the same for the line that contains “`disp('dong');`”.

Run the program. Instead of the output appearing, a green arrow should show up next to the breakpoint. This indicates the line of code that is about to be executed.

If you hover your mouse over variables in the code, you can view their current values. Try this with the **`i`** variable.

You’ll also notice the document buttons next to the run buttons have been activated.





```
for i = 1:8
    if mod(i, 2) == 0 % If remainder of i/2 is equal to 0
        disp('ding');
    else
        disp('dong');
    end
end
display('The witch is dead!');
```

The **step button** executes only the current line, then pauses again.
The **step-in button** steps into functions and loops.

The **continue button** resumes normal execution until another breakpoint is encountered. Pressing F5 also has the same effect.

Press the continue button, and observe the command window output. You'll also notice that your program has hit the other breakpoint. What values of "`i`" trigger the "ding" output and what values trigger "dong" ? Why?

Test Your New Skills

Create a new script called "Triangle.m" Using `for` loops and maybe `if` statements, write a program that asks the user for a size, and then generates a square of asterisks in the command window. For example if the user types in 5, then your script will output the following:

```
*  *  *  *  *
*  *  *  *  *
*  *  *  *  *
*  *  *  *  *
*  *  *  *  *
```

Hint: Remember that you can "nest" a **`for`** loop inside of another **`for`** loop.