

## MATLAB Marina – MATLAB Debugger

### Learning Objectives

1. Be able to use the MATLAB debugger to set breakpoints and debug programs

### Terms

syntax error, run time error, logic error, breakpoint, single step (step)

### MATLAB Functions, Keywords, and Operators

None

### MATLAB Debugger

MATLAB provides debugging capabilities via breakpoints and debug commands. Breakpoints are executable lines in the program or function that MATLAB can pause execution at while running the program.

MATLAB supports three types of breakpoints:

- Standard breakpoints where the program will pause at the specified line.
- Conditional breakpoints where the program will pause at the specified line only if the specified condition is met.
- Error breakpoints where the program will pause at a line only if a specified problem occurs (warning, error, NaN/infinite).

Standard breakpoints can be set by opening the program or function in the MATLAB editor and clicking on the line in the breakpoint alley (column just to right of line numbers). Lines that breakpoints can be set at are indicated by dashes. Standard breakpoints show up as grey (if program not saved) or red dots in the breakpoint alley. Conditional breakpoints can be set by selecting the breakpoint, right clicking and choosing Set/Modify Condition. You can then add a condition and hit ok to save the condition. Conditional breakpoints show up as yellow dots in the breakpoint alley. Breakpoints can be deleted in a similar manner. Breakpoints can also be set, cleared, enabled/disabled using the `Breakpoints` menu in the editor toolbar.

Breakpoints do not persist after you exit the MATLAB session.

Error breakpoints are not set at specific lines. Error breakpoints can be set by choosing the desired error handling, such as `Pause on Errors`, in the `Run` menu in the editor toolbar

When a program with breakpoints is run, MATLAB will enter debug mode (command line cursor changes from `>>` to `K>>`) and pause execution at the breakpoints. With execution paused, one can examine the current workspace and perform operations in the Command Window. Variable values can also be inspected by placing the mouse cursor on the variable. While program is paused during debugging, variable values can be changed and code can be modified although it is generally better to this after the debug run is finished so that changes can be tracked.

When in debug mode, the Run portion of the toolbar will show options for: Continue, Step, Step In, Step Out, and Run to Curser. Continue will resume execution until completion or until another breakpoint is encountered. Step will execute the current statement. Step In will step into a called function and execute the next statement. Step Out will execute the rest of the function, step out of the function and pause the program. Run to Curser executes the program to point where curser is. Debugging can be quit by choosing Quit Debugging.

### Debugging Example

Consider the program of Figure 1 that performs a running concatenation of positive numbers read from the user and creates a string.

---

```
% read in and save values until negative number entered
values = [];
newValue = input('Enter value (- number to stop): ');
while (newValue >= 0)
    values = [values, newValue];
    newValue = input('Enter value (- number to stop): ');
end

disp('Values are: '), disp(values);
```

---

Figure 1. Running Concatenation of Positive Numbers Read from user

For the running concatenation, the initial result is an empty vector and each new valid value is concatenated to the end of the current vector of values. Running concatenations can be used to create vectors of the same data type, for examples vectors of numbers or vectors of characters (strings).

Consider modifying the running concatenation of numbers of Figure 1 to read in and save characters read from the user. Using a sentinel as in the example of Figure 1 for numbers would be nice; what would be a good sentinel for reading in characters? Possibly the empty set as other characters even the special ones might be desirable to use (there is a special end of file (eof) character when performing file input and output). The program of Figure 2a is a first attempt at the running concatenation of characters.

---

```
% read in and save characters until terminating character read
characters = [];
newChar = input('Enter character(empty set to stop): ','s');
while (characters ~= [])
    character = [characters, newChar];
    newChar = input('Enter character(empty set to stop): ','s')
end

disp('Characters are: '), disp(characters);
```

---

Figure 2a. Attempt 1, Running Concatenation of Characters Read from User

A sample run of the program of Figure 2a is shown in Figure 2b.

```
Enter character(empty set to stop): a
Characters are:
>>
```

Figure 2b. Result of Running Program of Figure 2a

There are no syntax errors, but the program does not do what was hoped. One character (an a) is read in, but no characters are saved, and the program terminates before receiving the terminating indicator of the empty set.

The MATLAB debugger can be used to set breakpoints as shown in Figure 2c. Running the program with the breakpoints, MATLAB enters debug mode. The program is executed until the first breakpoint is reached (line 4). At the breakpoints one can examine workspace variables and see if the program is doing what was expected. To run to the next breakpoint, select the Step button in the Editor Toolstrip (or hit F10).

Figure 2d shows what appears in the MATLAB Command Window while stepping through the program. As the program is stepped through, execution stops at line 4, then line 5, then pauses for the user input, but never gets to line 7 or line 8. The while loop in the program is never executed. This is because the condition (`characters ~= []`) is never true. This condition always evaluates to the empty set (not true or false) and MATLAB does not treat the empty set as true.

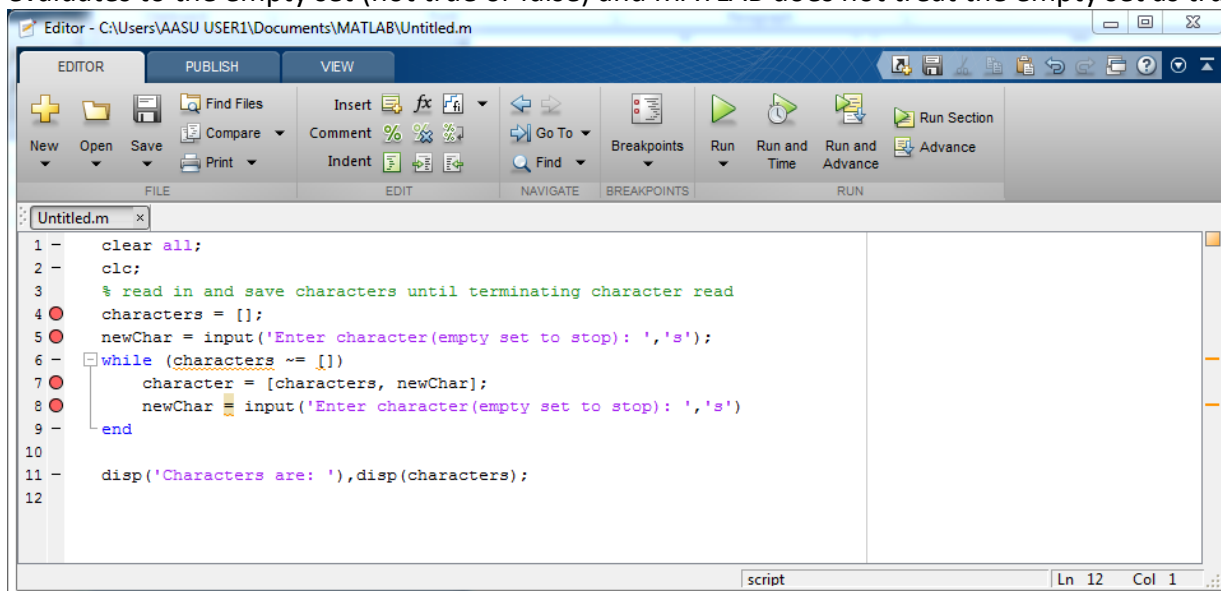
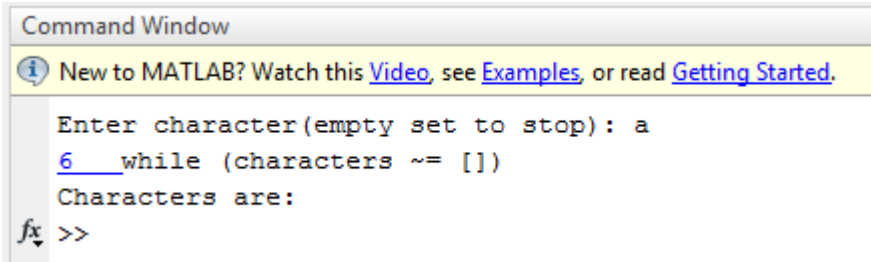


Figure 2c. Program with Breakpoints Set using MATLAB Editor



```
Command Window
New to MATLAB? Watch this Video, see Examples, or read Getting Started.
Enter character(empty set to stop): a
6 while (characters ~= [])
Characters are:
fx >>
```

Figure 2d. MATLAB Command Window During Debugging

Figure 2e shows the next attempt with the terminating character changed to a ~.

---

```
% read in and save characters until terminating character read
characters = [];
newChar = input('Enter character(~ to stop): ','s');
while (characters ~= ~)
    character = [characters, newChar];
    newChar = input('Enter character(empty set to stop): ','s')
end

disp('Characters are: '), disp(characters);
```

---

Figure 2e. Attempt 2, Running Concatenation of Characters Read from User

The program of 2e does not execute as it has a syntax error. The error message is shown in Figure 2f. This error message is not very descriptive, but line 6 is the line `while (characters ~= ~)`. The syntax error is due to not enclosing the ~ character in single quotes to indicate that it is a character.

---

```
Error: File: Untitled.m Line: 6 Column: 23
Unbalanced or unexpected parenthesis or bracket.
```

---

Figure 2f. Error Message from Running Program of Figure 2e

The program of Figure 2e has two additional logic errors in addition to the syntax error. The logic expression in the while statement should compare `newChar` not `characters`, which is the result vector, to the terminating character. The concatenating operation uses the variable `character` instead of the variable `characters` on the left side of the assignment. This will mean that each iteration, the value of `newChar` and `characters` will be concatenated but the result is saved in the variable `character` which is then not used for the next concatenation, i.e. nothing is saved but the last character read in before the terminating character.

There is also a style issue, in that the input line inside the while loop body is not terminated with a semicolon and thus the user input will be echoed during the execution of the script. The fully corrected program is shown in Figure 2g.

---

```
% read in and save characters until terminating character read
characters = [];
newChar = input('Enter character(~ to stop): ','s');
while (newChar ~= '~')
    characters = [characters, newChar];
    newChar = input('Enter character(empty set to stop): ','s');
end

disp('Characters are: '), disp(characters);
```

---


Figure 2g. Final Version of Running Concatenation of Characters Read from User

### Debugging Tips

Helpful tips for debugging:

- Syntax errors can be detected by running the program. MATLAB will identify the line number and type of error although the error message will not always clearly point to the error and some errors are actually caused by omitting statements prior to the line the error is indicated at (using a variable before it has been assigned a value).
- Logic errors can be identified using the MATLAB debugger and stepping through the program, examining the workspace variables at each step, and checking that the behavior is what it should be.
- Break the program/function up into sections. Implement and test each section separately. This helps to isolate errors to small sections of code.
- Make sure the sections are sequenced correctly. Sections that rely on operations in another section of the program must be placed after that section.
- When possible, test complex programs for simple cases with known solutions first. For example, when testing a program that approximates a cosine function using a Taylor series, test it first for known values of the cosine function (0, 45, 90 degrees).

Last modified January 24, 2022

 [MATLAB Marina](https://creativecommons.org/licenses/by-nc-sa/4.0/) is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.