

Lecture 4: Debugging and Polynomials

Math 98, Spring 2020

Reminders

Instructor: Andrew Shi

Login: !cmfmath98

Password: c@1parallel

Project:

- 1 Not due, but good practice.

Agenda

- 1 Debugging
 - ▶ See detailed agenda on next slide.
- 2 Polynomials
 - ▶ Evaluating them, Differentiating them, multiplying them, etc.

Debugging

- 1 How to avoid bugs (best programming practices)
 - ▶ Sections 2.7-2.8 of <http://www.sfu.ca/~wcs/ForGrads/ensc180spring2016f.pdf>
- 2 Warnings
- 3 Breakpoints and Step
- 4 Step in/out
- 5 Run Options

Will closely follow the MATLAB documentation

https://www.mathworks.com/help/matlab/matlab_prog/debugging-process-and-features.html.

Incremental Development

When you start writing scripts that are more than a few lines, you might find yourself spending more and more time debugging. The more code you write before you start debugging, the harder it is to find the problem.

Incremental development is a way of programming that tries to minimize the pain of debugging.

Incremental Development: Three Steps

The fundamental steps of incremental debugging are:

- 1 Always start with a working program. If you have an example from a book or a program you wrote that is similar to what you are working on, start with that. Otherwise, start with something you know is correct, like $x = 5$. Run the program and confirm that you are running the program you think you are running. This step is important, because in most environments there are lots of little things that can trip you up when you start a new project. Get them out of the way so you can focus on programming.
- 2 Make one small, testable change at a time. A “testable” change is one that displays something on the screen (or has some other effect) that you can check. Ideally, you should know what the correct answer is, or be able to check it by performing another computation.
- 3 Run the program and see if the change worked. If so, go back to Step 2. If not, you will have to do some debugging, but if the change you made was small, it shouldn't take long to find the problem.

Unit Testing

In large software projects, **unit testing** is the process of testing software components in isolation before putting them together.

The programs we have seen so far are not big enough to need unit testing, but the same principle applies when you are working with a new function or a new language feature for the first time. You should test it in isolation before you put it into your program.

Unit Testing: Example

For example, suppose you know that x is the sine of some angle and you want to find the angle. You find the MATLAB function `asin`, and you are pretty sure it computes the inverse sine function. Pretty sure is not good enough; you want to be very sure.

Since we know $\sin(0) = 0$, we could try:

```
>> asin(0)
ans = 0
```

which is correct. We also know that \sin of 90° is 1, so if we try `asin(1)` we expect the answer 90, right?

```
>> asin(1)
ans = 1.5708
```

What's going on here?

Unit Testing: Example (Cont)

Oops. We forgot that the trig functions in MATLAB work in radians, not degrees. So the correct answer is $\frac{\pi}{2}$, which we can confirm by dividing through by π :

```
>> asin(1)/pi  
ans = 0.5000
```

With this kind of unit testing, you are not really checking for errors in MATLAB, you are checking your understanding. If you make an error because you are confused about how MATLAB works, it might take a long time to find, because when you look at the code, it looks right.

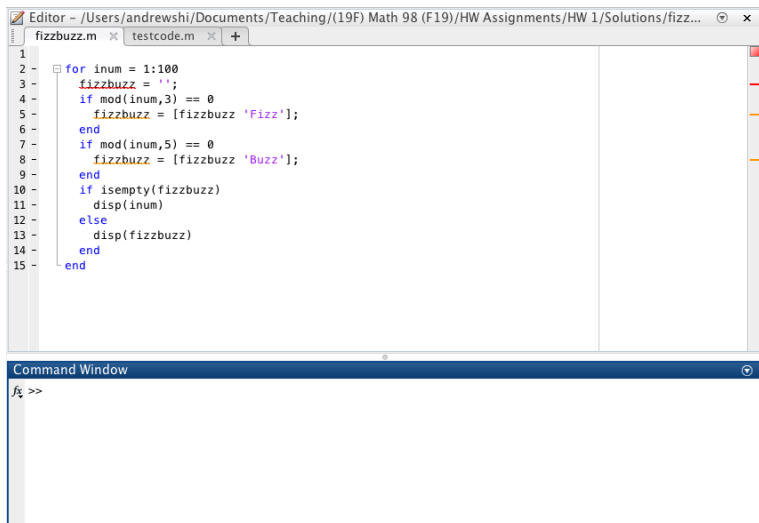
The worst bugs aren't in your code; they are in your head

Debugging in four acts

- **Reading:** Examine your code, read it back to yourself, and check that it means what you meant to say.
- **Running:** Experiment by making changes and running different versions. Often if you display the right thing at the right place in the program, the problem becomes obvious, but sometimes you have to spend some time to build scaffolding.
- **Ruminating:** Take some time to think! What kind of error is it: syntax, runtime, logical? What information can you get from the error messages, or from the output of the program? What kind of error could cause the problem you're seeing? What did you change last, before the problem appeared?
- **Retreating:** At some point, the best thing to do is back off, undoing recent changes, until you get back to a program that works, and that you understand. Then you can start rebuilding.

Warnings

Here's an implementation of `fizzbuzz.m` from HW1. Do you see the warnings? (red underlines).



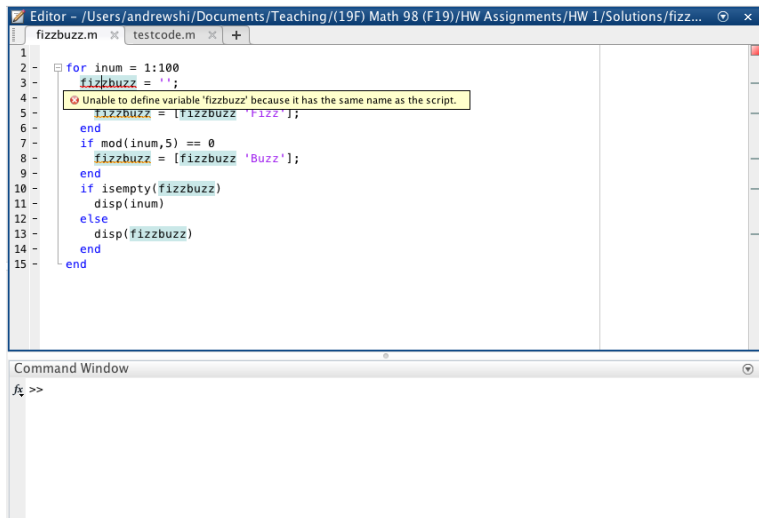
The screenshot shows the MATLAB Editor window with a script named `fizzbuzz.m`. The script contains the following code:

```
1 for inum = 1:100
2     fizzbuzz = '';
3     if mod(inum,3) == 0
4         fizzbuzz = [fizzbuzz 'Fizz'];
5     end
6     if mod(inum,5) == 0
7         fizzbuzz = [fizzbuzz 'Buzz'];
8     end
9     if isempty(fizzbuzz)
10        disp(inum)
11    else
12        disp(fizzbuzz)
13    end
14 end
15 end
```

The Command Window at the bottom shows the prompt `>>` and is currently empty. The MATLAB interface includes a toolbar at the bottom right with icons for navigation and search.

Warnings

If I hover over the squiggly red line, it tells me the issue. Red warnings need to be addressed for the code to run (things like syntax error).



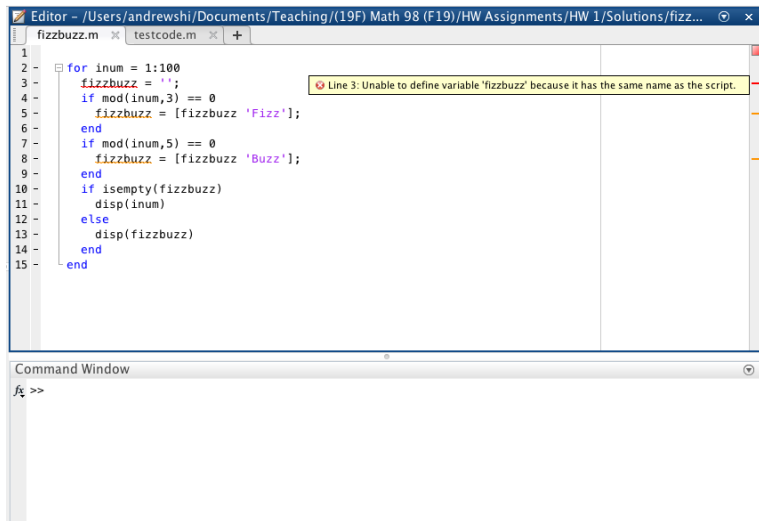
The screenshot shows the MATLAB Editor interface. The main window displays a script named 'fizzbuzz.m' with the following code:

```
1 for inum = 1:100
2     fizzbuzz = '';
3     [fizzbuzz 'fizz'] = [fizzbuzz 'fizz'];
4     end
5     if mod(inum,5) == 0
6         fizzbuzz = [fizzbuzz 'Buzz'];
7     end
8     if isempty(fizzbuzz)
9         disp(inum)
10    else
11        disp(fizzbuzz)
12    end
13 end
14
15
```

A red squiggly line is under the variable 'fizzbuzz' on line 3. A yellow tooltip box is displayed over this line, containing the warning: "Unable to define variable 'fizzbuzz' because it has the same name as the script." The Command Window at the bottom shows the prompt 'fz >>'.

Warnings

You can hover over the line on the right and it will give you the same warning (with line number)



The image shows a MATLAB Editor window with a script named 'fizzbuzz.m'. The script contains a for loop that iterates from 1 to 100. On line 3, the variable 'fizzbuzz' is assigned an empty string. A warning message is displayed on the right side of the editor, pointing to line 3: 'Line 3: Unable to define variable 'fizzbuzz' because it has the same name as the script.' The Command Window at the bottom shows the prompt 'fx >>'.

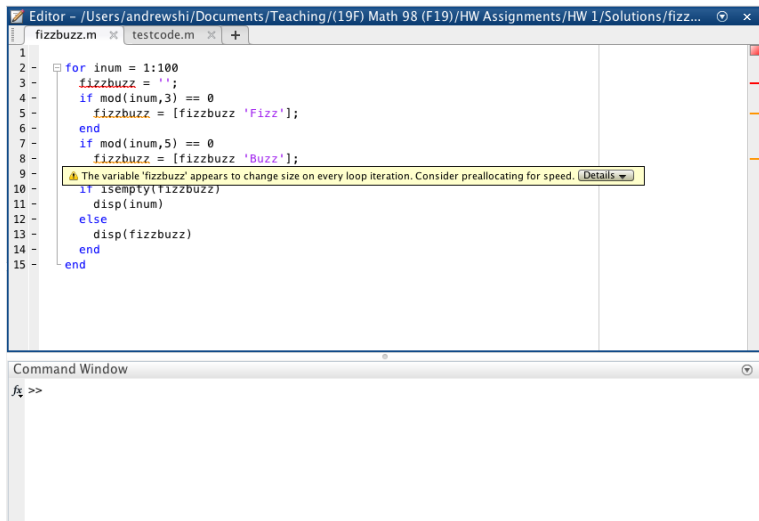
```
1 for inum = 1:100
2     fizzbuzz = '';
3     if mod(inum,3) == 0
4         fizzbuzz = [fizzbuzz 'Fizz'];
5     end
6     if mod(inum,5) == 0
7         fizzbuzz = [fizzbuzz 'Buzz'];
8     end
9     if isempty(fizzbuzz)
10        disp(inum)
11    else
12        disp(fizzbuzz)
13    end
14 end
15 end
```

Line 3: Unable to define variable 'fizzbuzz' because it has the same name as the script.

Command Window
fx >>

Warnings

The orange warnings aren't fatal, and usually indicate some inefficiency in your implementation. But you should still try to address them.



The screenshot shows the MATLAB Editor window with a file named 'fizzbuzz.m' open. The code is as follows:

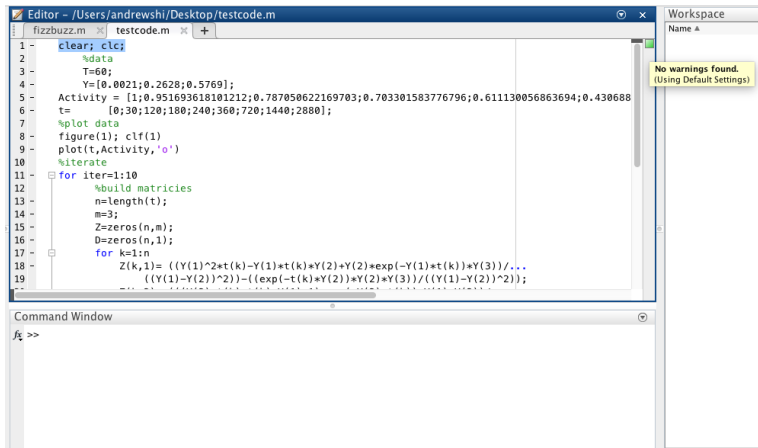
```
1
2 for inum = 1:100
3     fizzbuzz = '';
4     if mod(inum,3) == 0
5         fizzbuzz = [fizzbuzz 'Fizz'];
6     end
7     if mod(inum,5) == 0
8         fizzbuzz = [fizzbuzz 'Buzz'];
9
10    if isempty(fizzbuzz)
11        disp(inum)
12    else
13        disp(fizzbuzz)
14    end
15 end
```

A warning box is displayed over line 9, containing the text: "The variable 'fizzbuzz' appears to change size on every loop iteration. Consider preallocating for speed. Details".

Below the editor is the Command Window, which shows the prompt `fz >>` and is currently empty.

Warnings

Here's some random code I found on the internet that someone posted to get help on. Note that there are no warnings in the code (green box).



The screenshot shows a MATLAB editor window titled "Editor - /Users/andrewshi/Desktop/testcode.m". The code in the editor is as follows:

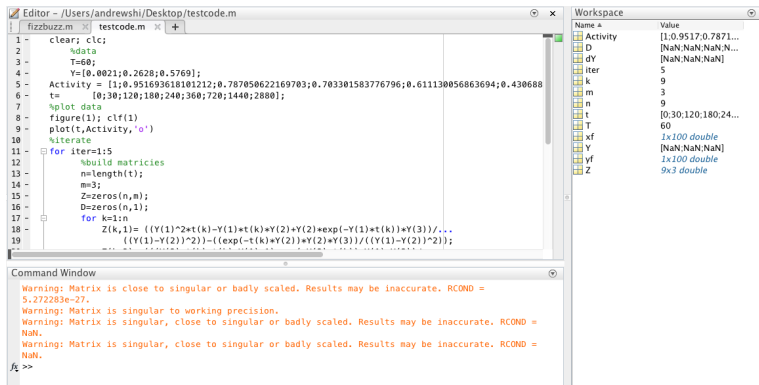
```
1 clear; clc;
2 %data
3 T=60;
4 Y=[0.0021;0.2628;0.5769];
5 Activity = [1;0.951693618101212;0.787050622169703;0.703301583776796;0.611130056863694;0.430688
6 t= [0;30;120;180;240;360;720;1440;2880];
7 %plot data
8 figure(1); clf(1)
9 plot(t,Activity,'o')
10 %iterate
11 for iter=1:10
12     %build matrices
13     n=length(t);
14     m=3;
15     Z=zeros(n,m);
16     D=zeros(n,1);
17     for k=1:n
18         Z(k,1)= ((Y(1)^2*t(k)-Y(1)*t(k)*Y(2)+Y(2)*exp(-Y(1)*t(k))*Y(3))/...
19                 ((Y(1)-Y(2))^2)-{(exp(-t(k)*Y(2))*Y(2)*Y(3))/(Y(1)-Y(2))^2);
```

A yellow tooltip message is displayed over the code, stating: "No warnings found. (Using Default Settings)".

The Command Window at the bottom shows the prompt `>>`.

Warnings

But when you run it, all these warnings comes out. This warning in particular shows up when you try to invert a matrix that is nearly singular. Note all the NaNs he got in his results.



The screenshot shows a MATLAB Editor window with a script named testcode.m. The script defines a matrix Y and iteratively updates it. The Command Window shows the following warnings:

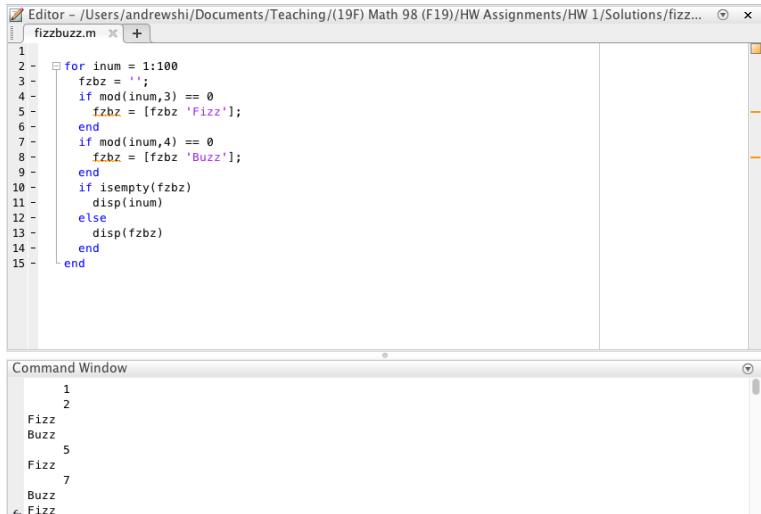
```
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND = 5.272283e-27.  
Warning: Matrix is singular to working precision.  
Warning: Matrix is singular, close to singular or badly scaled. Results may be inaccurate. RCOND = NaN.  
Warning: Matrix is singular, close to singular or badly scaled. Results may be inaccurate. RCOND = NaN.
```

The Workspace window shows the following variables and their values:

Name	Value
Activity	[1; 0.9517; 0.7871; ...]
D	[NaN; NaN; NaN; NaN; ...]
dY	[NaN; NaN; NaN]
iter	5
k	9
m	3
n	9
t	[0; 30; 120; 180; 240; ...]
T	60
xf	1x100 double
Y	[NaN; NaN; NaN]
yf	1x100 double
Z	9x3 double

Breakpoints

I fixed my fatal error but not the other one (because I'm lazy) and I run fizzbuzz code. The output looks wrong! (remember buzz means divisible by 5).



The screenshot shows a MATLAB Editor window with a script named 'fizzbuzz.m'. The script is as follows:

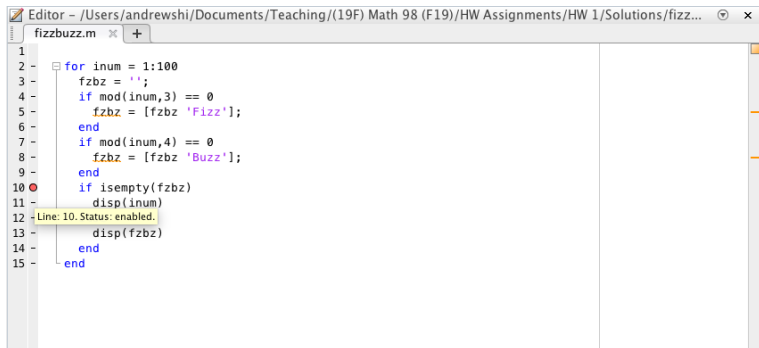
```
1 for inum = 1:100
2     fzbz = '';
3     if mod(inum,3) == 0
4         fzbz = [fzbz 'Fizz'];
5     end
6     if mod(inum,4) == 0
7         fzbz = [fzbz 'Buzz'];
8     end
9     if isempty(fzbz)
10        disp(inum)
11    else
12        disp(fzbz)
13    end
14 end
15
```

The Command Window below shows the output of the script:

```
1
2
Fizz
Buzz
5
Fizz
7
Buzz
Fizz
```


Breakpoints

If I click one of the lines on the left it will create this red dot. This dot is called a breakpoint. I can easily remove it by clicking on it again.



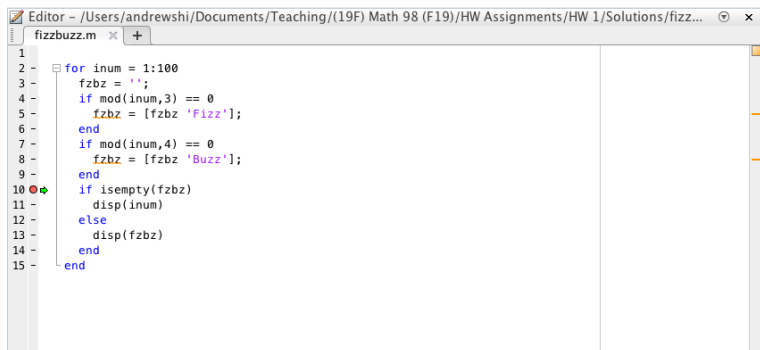
The screenshot shows a MATLAB editor window titled "Editor - /Users/andrewshi/Documents/Teaching/(19F) Math 98 (F19)/HW Assignments/HW 1/Solutions/fizz...". The script file is named "fizzbuzz.m". The code is as follows:

```
1
2 - for inum = 1:100
3   fzbz = '';
4   if mod(inum,3) == 0
5     fzbz = [fzbz 'Fizz'];
6   end
7   if mod(inum,4) == 0
8     fzbz = [fzbz 'Buzz'];
9   end
10 ● if isempty(fzbz)
11     disp(inum)
12 - Line: 10. Status: enabled.
13     disp(fzbz)
14   end
15 - end
```

A red dot is placed on the left margin of line 10, indicating an enabled breakpoint. A tooltip below the dot reads "Line: 10. Status: enabled.".

Breakpoints

If I run my code again, it will stop the first time I hit the line with a breakpoint. The green arrow tells me where I am now.



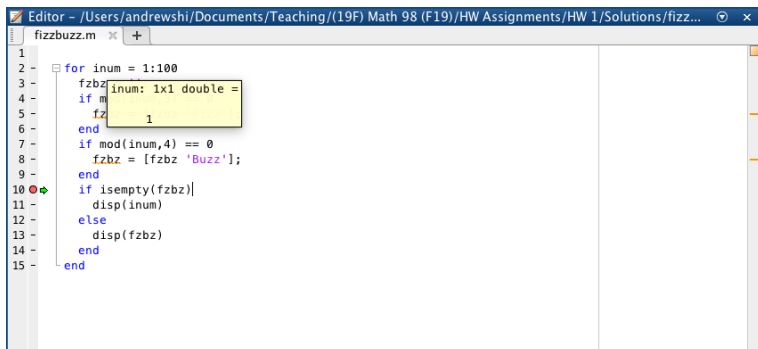
The screenshot shows a MATLAB editor window titled "Editor - /Users/andrewshi/Documents/Teaching/(19F) Math 98 (F19)/HW Assignments/HW 1/Solutions/fizz...". The script file is named "fizzbuzz.m". The code is as follows:

```
1  
2 - for inum = 1:100  
3 -     fzbz = '';  
4 -     if mod(inum,3) == 0  
5 -         fzbz = [fzbz 'Fizz'];  
6 -     end  
7 -     if mod(inum,4) == 0  
8 -         fzbz = [fzbz 'Buzz'];  
9 -     end  
10 - ●▶ if isempty(fzbz)  
11 -     disp(inum)  
12 -     else  
13 -     disp(fzbz)  
14 -     end  
15 - end
```

A red circle with a green arrow is positioned to the left of line 10, indicating a breakpoint. The editor interface includes a toolbar with various icons for file operations and a scroll bar on the right side.

Breakpoints

I can hover over variables to see what their current value is right now at the breakpoint.

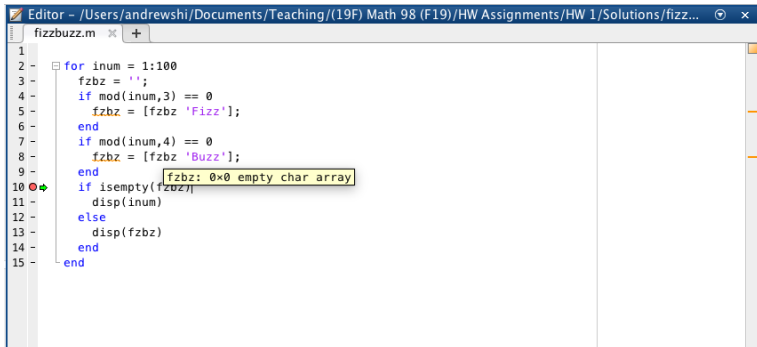


The screenshot shows a MATLAB editor window titled "Editor - /Users/andrewshi/Documents/Teaching/(19F) Math 98 (F19)/HW Assignments/HW 1/Solutions/fizz...". The script "fizzbuzz.m" is open, and a red arrow breakpoint is set at line 10. A tooltip is displayed over the variable "inum" on line 3, showing its current value as "inum: 1x1 double = 1". The script code is as follows:

```
1 -  
2 - for inum = 1:100  
3 -     fzbz  
4 -     if mod(inum,3) == 0  
5 -         fzbz = [fzbz 'Fizz'];  
6 -     end  
7 -     if mod(inum,4) == 0  
8 -         fzbz = [fzbz 'Buzz'];  
9 -     end  
10 -    if isempty(fzbz)  
11 -        disp(inum)  
12 -    else  
13 -        disp(fzbz)  
14 -    end  
15 - end
```

Breakpoints

I can hover over variables to see what their current value is right now at the breakpoint.

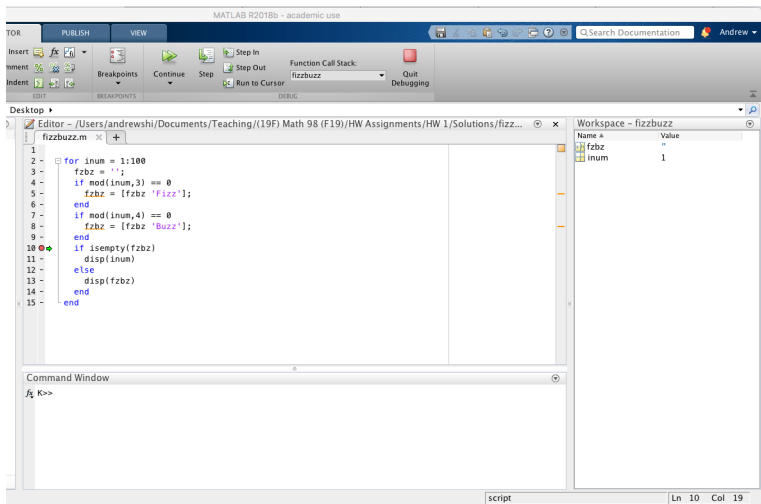


```
Editor - /Users/andrewshi/Documents/Teaching/(19F) Math 98 (F19)/HW Assignments/HW 1/Solutions/fizz... x
fizzbuzz.m x +
1
2 - for inum = 1:100
3   fzbx = '';
4   if mod(inum,3) == 0
5     fzbx = [fzbx 'Fizz'];
6   end
7   if mod(inum,4) == 0
8     fzbx = [fzbx 'Buzz'];
9   end
10  if isempty(fzbx)
11     disp(inum)
12  else
13     disp(fzbx)
14  end
15  end
```

Tooltip: fzbx: 0x0 empty char array

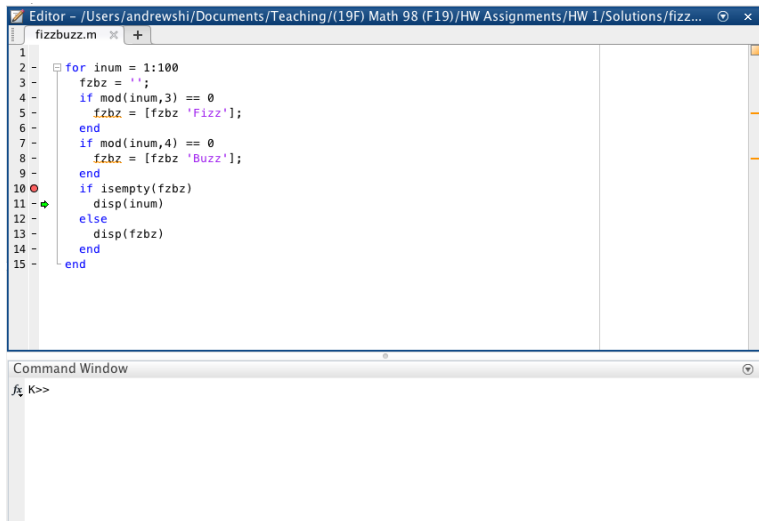
Breakpoints

We are now in debug mode. We could always exit by hitting that red button “Quit Debugging”. Note that K that isn’t usually there in the command window.



Breakpoints

If you want to go another line further, hit step.

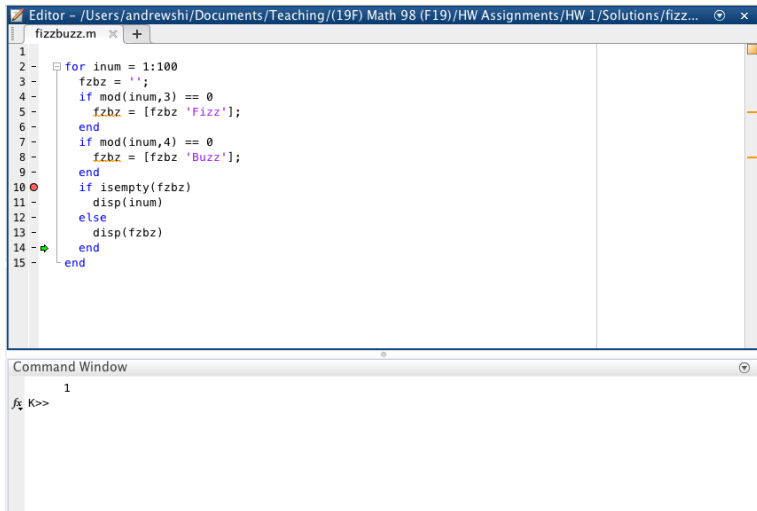


```
Editor - /Users/andrewshi/Documents/Teaching/(19F) Math 98 (F19)/HW Assignments/HW 1/Solutions/fizz...
fizzbuzz.m
1
2 - for inum = 1:100
3   fzbz = '';
4   if mod(inum,3) == 0
5     fzbz = [fzbz 'Fizz'];
6   end
7   if mod(inum,4) == 0
8     fzbz = [fzbz 'Buzz'];
9   end
10  if isempty(fzbz)
11  → disp(inum)
12  else
13    disp(fzbz)
14  end
15  end

Command Window
fx K>>
```

Breakpoints

If you want to go another line further, hit step. Note that we just executed line 11 so something displayed in the command window. (Question: why did we skip from line 11 to 14?)



The screenshot shows the MATLAB Editor window with a script named 'fizzbuzz.m'. The script contains a loop from 1 to 15. A red circle indicates a breakpoint is set at line 11. The Command Window shows the output of the script, with the number '1' displayed, indicating that the script has executed line 11 and then jumped to line 14.

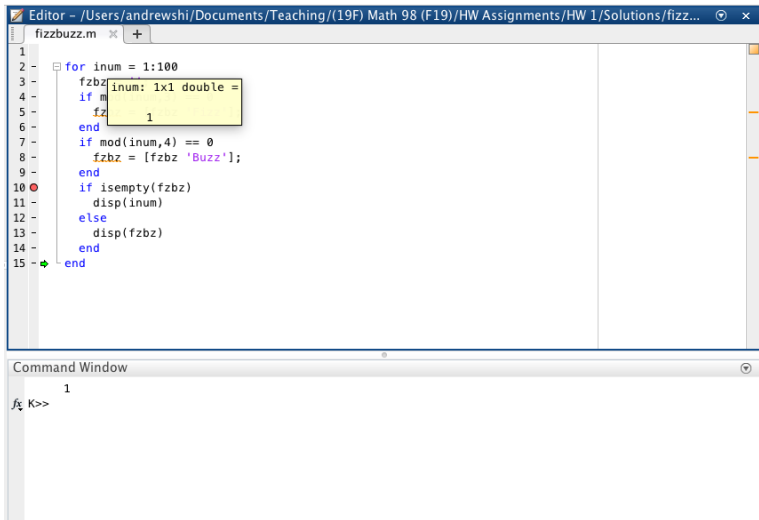
```
1  
2 for inum = 1:100  
3     fzbz = '';  
4     if mod(inum,3) == 0  
5         fzbz = [fzbz 'Fizz'];  
6     end  
7     if mod(inum,4) == 0  
8         fzbz = [fzbz 'Buzz'];  
9     end  
10    if isempty(fzbz)  
11        disp(inum)  
12    else  
13        disp(fzbz)  
14    end  
15 end
```

Command Window

```
1  
K>>
```

Breakpoints

We are at the end of the loop. This is the last point `inum = 1`.



The image shows a MATLAB Editor window with a script named `fizzbuzz.m`. The script contains a `for` loop from `inum = 1:100`. A red circle breakpoint is set at line 15, which is the `end` of the loop. A yellow tooltip is visible over the `end` keyword, displaying the variable `inum` with a value of `1`. Below the editor is the Command Window, which shows the output `1` from the `disp(inum)` statement on line 11.

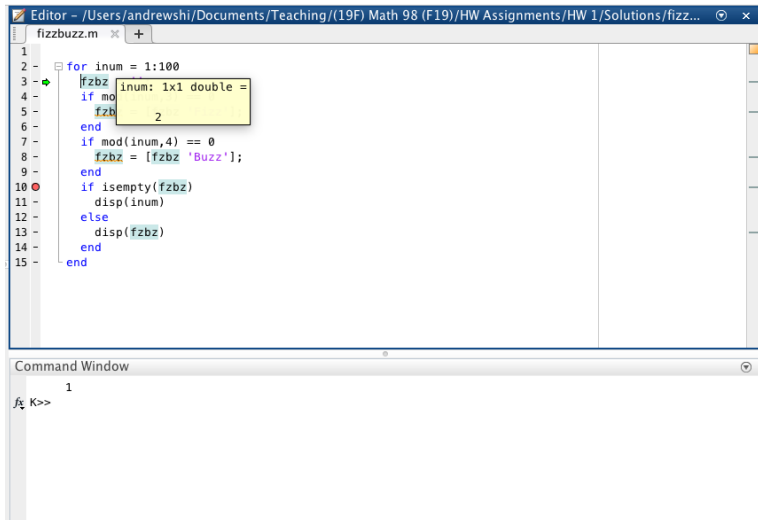
```
1 for inum = 1:100
2     fzbz = [];
3     if mod(inum,4) == 0
4         fzbz = [fzbz 'Buzz'];
5     end
6     if isempty(fzbz)
7         disp(inum)
8     else
9         disp(fzbz)
10    end
11 end
12
```

Command Window

```
1
fx K>>
```


Breakpoints

And now `inum = 2`.



The image shows a MATLAB Editor window with a script named `fizzbuzz.m`. The script contains the following code:

```
1 for inum = 1:100
2     fzbz
3     if mod(inum,2) == 0
4         fzbz = [fzbz 'Buzz'];
5     end
6     if mod(inum,3) == 0
7         fzbz = [fzbz 'Fizz'];
8     end
9     if isempty(fzbz)
10        disp(inum)
11    else
12        disp(fzbz)
13    end
14 end
15
```

A breakpoint is set at line 3, indicated by a green diamond. A yellow tooltip is visible over the `inum` variable in the `if` statement, displaying the text `inum: 1x1 double = 2`. The Command Window at the bottom shows the current value of `inum` as 1, with the prompt `K>>`.

Breakpoints

If you hit continue, you will keep running until the next breakpoint.

The image shows the MATLAB R2018b - academic use interface. The top toolbar includes buttons for 'PUBLISH', 'VIEW', 'Breakpoints', 'Continue', 'Step', 'Step In', 'Step Out', 'Run to Cursor', 'Function Call Stack' (showing 'fizzbuzz'), and 'Quit Debugging'. The main editor window displays a script named 'fizzbuzz.m' with the following code:

```
1 for inum = 1:100
2     fzbz = [];
3     for f = 1:2
4         if mod(inum,f) == 0;
5             fzbz = [fzbz num2str(inum) ' '];
6         end
7     end
8     if mod(inum,4) == 0
9         fzbz = [fzbz 'Buzz'];
10    end
11    if isempty(fzbz)
12        disp(inum)
13    else
14        disp(fzbz)
15    end
end
```

A yellow tooltip is visible over the code, displaying 'inum: 1x1 double = 2'. A red arrow points to line 10. The Command Window at the bottom shows the prompt 'K>>' and the number '1'.

Breakpoints

If you hit continue, you will keep running until the next breakpoint.

The image shows the MATLAB R2018b - academic use interface. The top toolbar includes buttons for 'Breakpoints', 'Continue', 'Step', 'Step In', 'Step Out', 'Run to Cursor', and 'Quit Debugging'. The 'Function Call Stack' shows 'fizzbuzz'. The main editor window displays the following code:

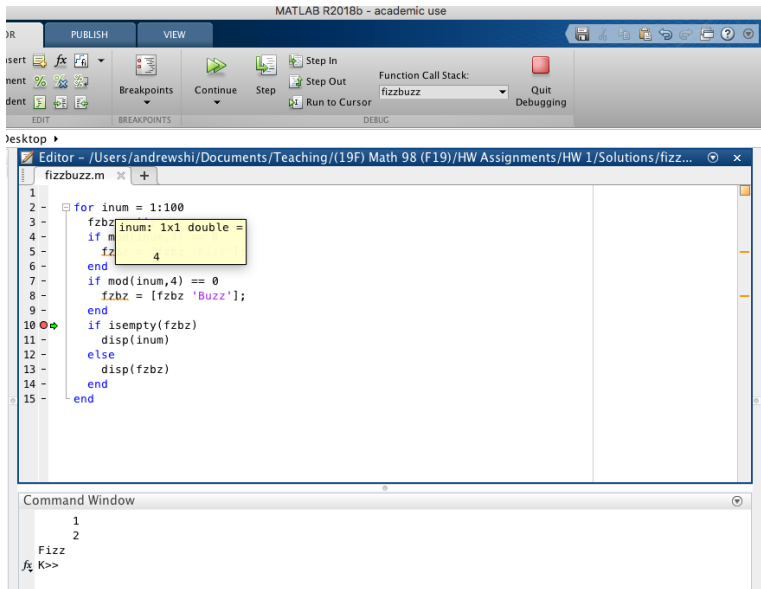
```
1  
2 - for inum = 1:100  
3 -     fzbz = []  
4 -     if mod(inum,3) == 0  
5 -         fzbz = [fzbz inum];  
6 -     end  
7 -     if mod(inum,4) == 0  
8 -         fzbz = [fzbz 'Buzz'];  
9 -     end  
10 - if isempty(fzbz)  
11 -     disp(inum)  
12 - else  
13 -     disp(fzbz)  
14 - end  
15 - end
```

A yellow tooltip is visible over line 4, displaying: 'inum: 1x1 double = 3'. The Command Window at the bottom shows the output of the script:

```
1  
2  
fx K>>
```

Breakpoints

If you hit continue, you will keep running until the next breakpoint.



The image shows the MATLAB R2018b - academic use interface. The top toolbar includes buttons for PUBLISH, VIEW, Breakpoints, Continue, Step, Step In, Step Out, Run to Cursor, and Quit Debugging. The Function Call Stack shows 'fizzbuzz'. The main editor window displays the code for 'fizzbuzz.m' with a breakpoint set on line 10. A tooltip shows the current value of 'inum' as 4. The Command Window shows the output of the program.

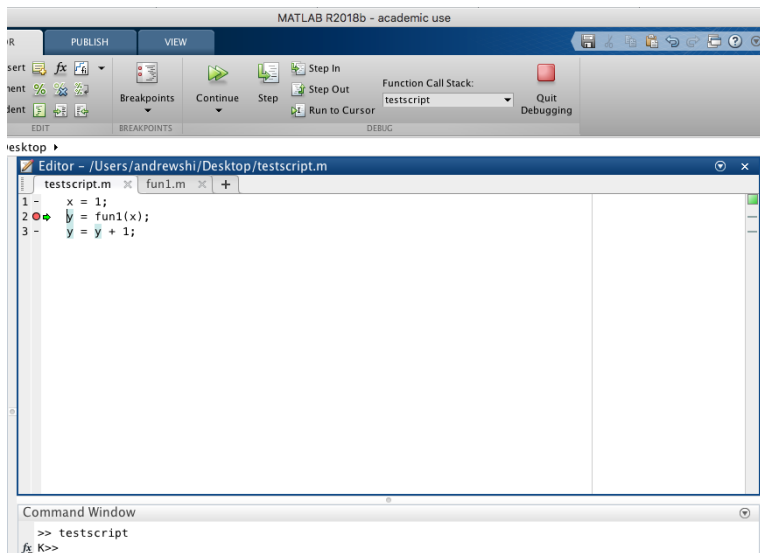
```
1  
2 for inum = 1:100  
3     fzbz = [];  
4     if mod(inum,3) == 0  
5         fzbz = [fzbz 'Fizz'];  
6     end  
7     if mod(inum,4) == 0  
8         fzbz = [fzbz 'Buzz'];  
9     end  
10    if isempty(fzbz)  
11        disp(inum)  
12    else  
13        disp(fzbz)  
14    end  
15 end
```

Command Window

```
1  
2  
Fizz  
K>>
```

Step In/Out

I've written a script that calls a function `fun1.m`. I think something bad is going on in there so I put a breakpoint there and run.



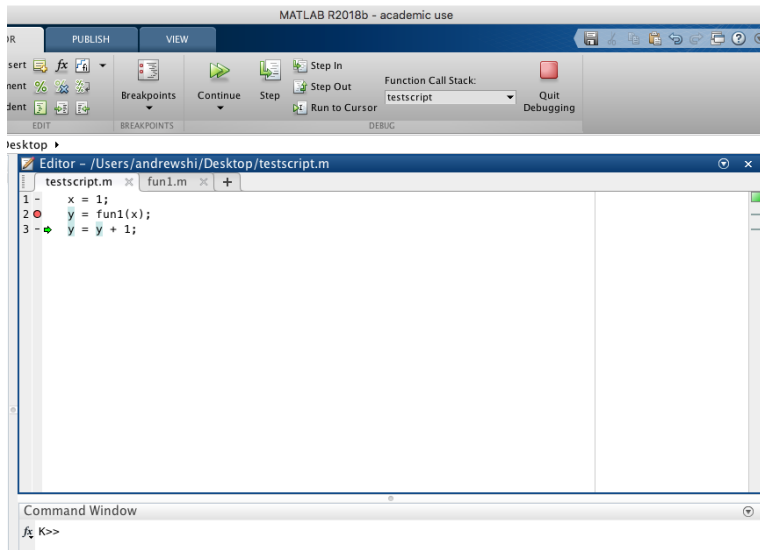
The image shows the MATLAB R2018b debugging interface. The top toolbar includes buttons for 'Continue', 'Step', 'Step In', 'Step Out', 'Run to Cursor', and 'Quit Debugging'. The 'Function Call Stack' dropdown is set to 'testscript'. Below the toolbar, the 'Desktop' tab is active, showing the 'Editor' window for `/Users/andrewshi/Desktop/testscript.m`. The editor displays the following code:

```
1 - x = 1;  
2 - y = fun1(x);  
3 - y = y + 1;
```

A red circle with a white arrow (breakpoint) is set on line 2. The 'Command Window' at the bottom shows the command `>> testscript` and the prompt `f> K>>`.

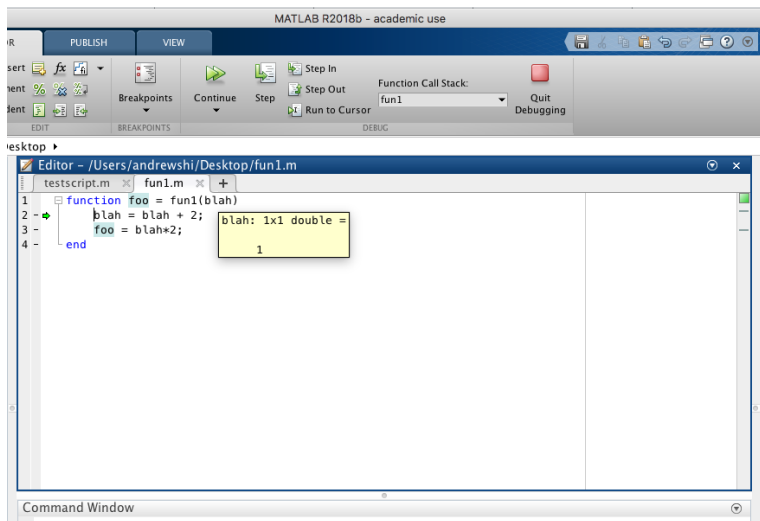
Step In/Out

But if I just step like before it runs function `fun1.m` and I have no insight as to what happened inside.



Step In/Out

What I really want to do is Step In, so now I'm inside the function call. Note it passed in the value `x` from outside as the value for `blah` that `fun1.m` takes.



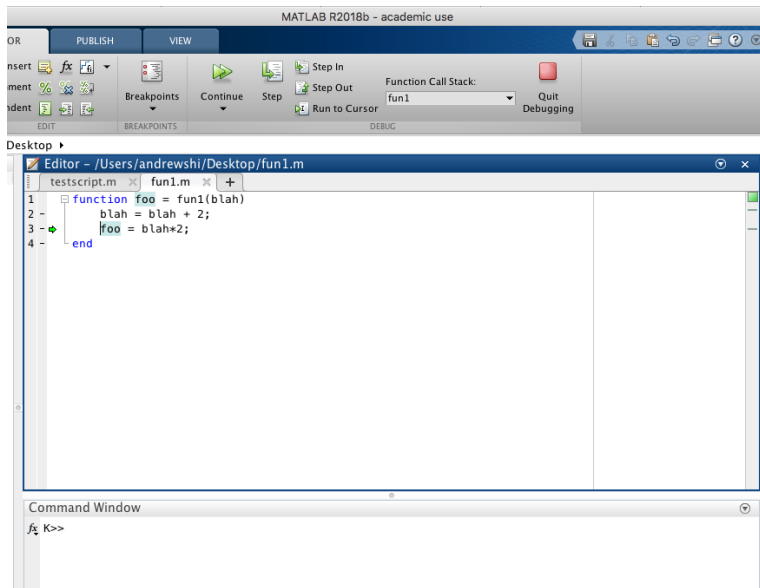
The image shows the MATLAB R2018b - academic use interface. The top toolbar includes buttons for PUBLISH, VIEW, and various debugging actions: Step In, Step Out, Continue, Step, Run to Cursor, and Quit Debugging. The Function Call Stack shows 'fun1' as the current function. The Editor window displays the code for 'fun1.m' with a breakpoint set at line 2. A tooltip shows the variable 'blah' is a 1x1 double with the value 1. The Command Window is visible at the bottom.

```
function foo = fun1(blah)
    blah = blah + 2;
    foo = blah*2;
end
```

Command Window

Step In/Out

In here I can keep stepping just like before....



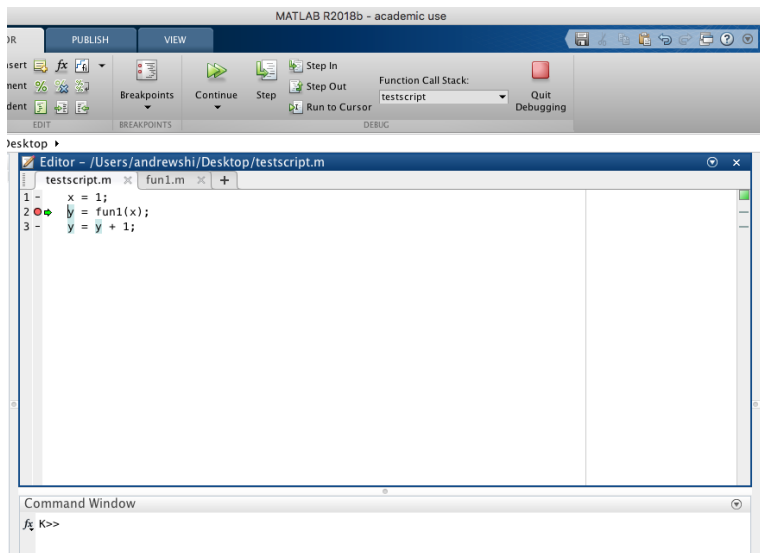
The image shows the MATLAB R2018b - academic use interface. The top toolbar includes buttons for 'Step In', 'Step Out', 'Continue', and 'Run to Cursor'. The 'Function Call Stack' shows 'fun1' as the current function. The main editor window displays the following code:

```
1 function foo = fun1(blah)
2     blah = blah + 2;
3     foo = blah+2;
4 end
```

The Command Window at the bottom shows the prompt `K>>`.

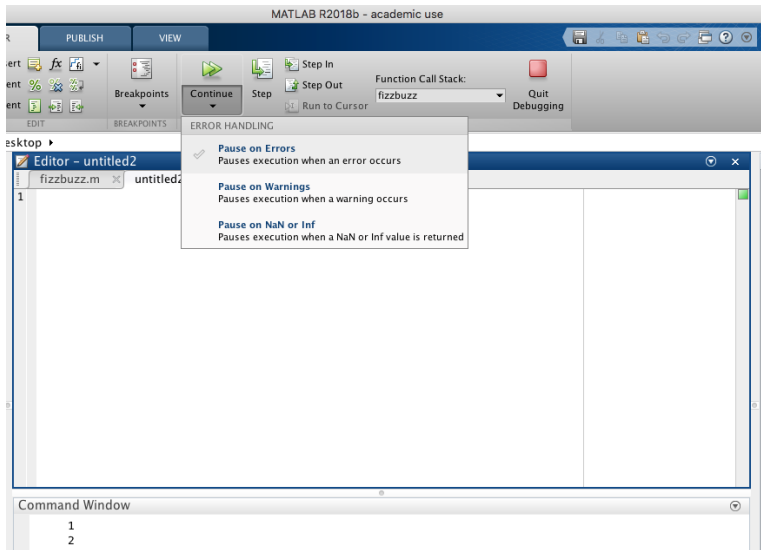
Step In/Out

If I want to leave the function I can click Step Out. Note that a value hasn't been assigned to `y` yet, so I didn't finish running `fun1`, I just left it.



Run Options

There are three Run Options you can choose, and your code will pause at the line one of these designated events occurs.



Print Statements

Another widely used technique is to put print statements in your code so you can monitor certain variables.

Debugging can take some time to get used to and can be very frustrating.

Polynomials

Many of the main algorithms in Math 128a involve replacing a general function $f(x)$ with an approximating polynomial $P(x)$.

In MATLAB, if you want to define a polynomial (more generally a function) and evaluate it, we use anonymous function handles.

```
>> f = @(x) x^2 + 2*x + 4; f(2)
ans = 12
```

We can also represent a polynomial with a vector.

```
>> p = [1, 2, 4]; polyval(p, 2)
ans = 12
```

Polynomials

Can you figure out how to differentiate a polynomial?

```
>> p = [1, 2, 4]; diffp = SOMEFUNCTION(p)
ans = 2 2
```

How about multiplying two polynomials together?

```
>> p1 = [1, 3]; p2 = [1, 1]; p1p2prod = SOMEFUNCTION(p1, p2)
ans = 1 4 3
```

How do you fit a polynomial of degree n to $n + 1$ points?

```
>> pts = [1, 2; 3, 3; 4, 5];
.....
```

Exercise: polycrazy.m

Write a function:

$$[\text{max}] = \text{polycrazy}(f, n, [a, b])$$

that does the following:

- Takes in a function handle f and evaluates it at n equispaced points on the interval $[a, b]$.
- Fits a degree $n - 1$ polynomial interpolant to those n points.
- Returns the maximum value of that interpolant on the interval.