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.

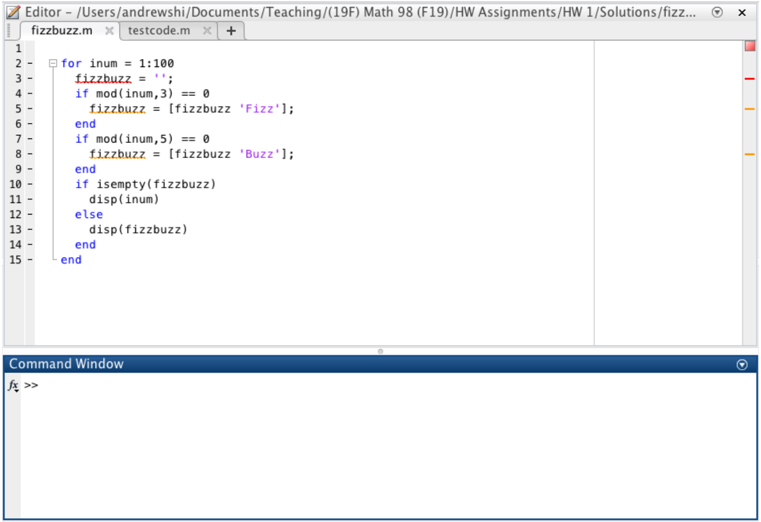
> The worst bugs aren't in your code; they are in your head

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 starting rebuilding.

Warnings

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



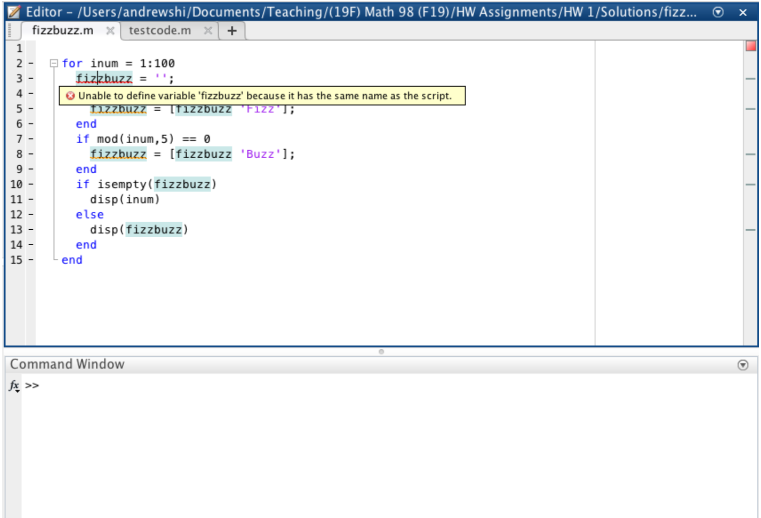
Editor – /Users/andrewshi/Documents/Teaching/(19F) Math 98 (F19)/HW Assignments/HW 1/Solutions/fizz...
fizzbuzz.m
testcode.m
+
Command Window

```matlab

for inum = 1:100
  fizzbuzz = '';
  if mod(inum,3) == 0
    fizzbuzz = [fizzbuzz 'Fizz'];
  end
  if mod(inum,5) == 0
    fizzbuzz = [fizzbuzz 'Buzz'];
  end
  if isempty(fizzbuzz)
    disp(inum)
  else
    disp(fizzbuzz)
  end
end
```

```
fx >>
```

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).



Editor – /Users/andrewshi/Documents/Teaching/(19F) Math 98 (F19)/HW Assignments/HW 1/Solutions/fizz...

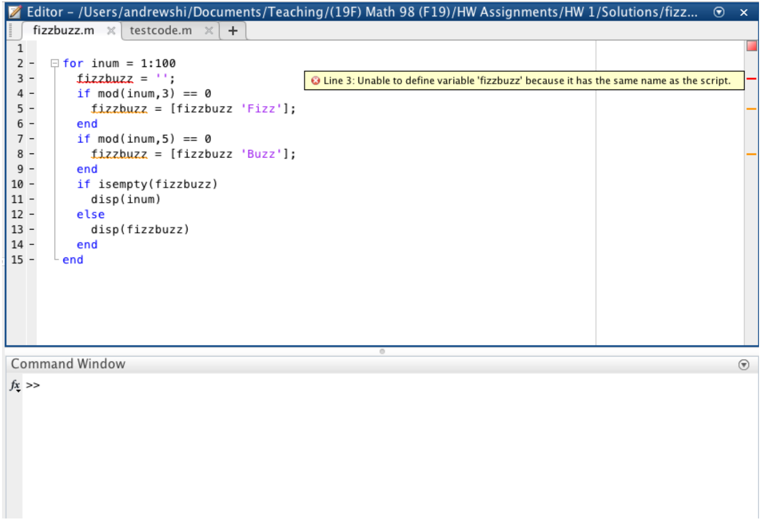
```

for inum = 1:100
  fizzbuzz = '';

    fizzbuzz = [fizzbuzz 'Fizz'];
  end
  if mod(inum,5) == 0
    fizzbuzz = [fizzbuzz 'Buzz'];
  end
  if isempty(fizzbuzz)
    disp(inum)
  else
    disp(fizzbuzz)
  end
end
```

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

```
fx >>
```

Warnings

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



Editor – /Users/andrewshi/Documents/Teaching/(19F) Math 98 (F19)/HW Assignments/HW 1/Solutions/fizz...
fizzbuzz.m
testcode.m
+

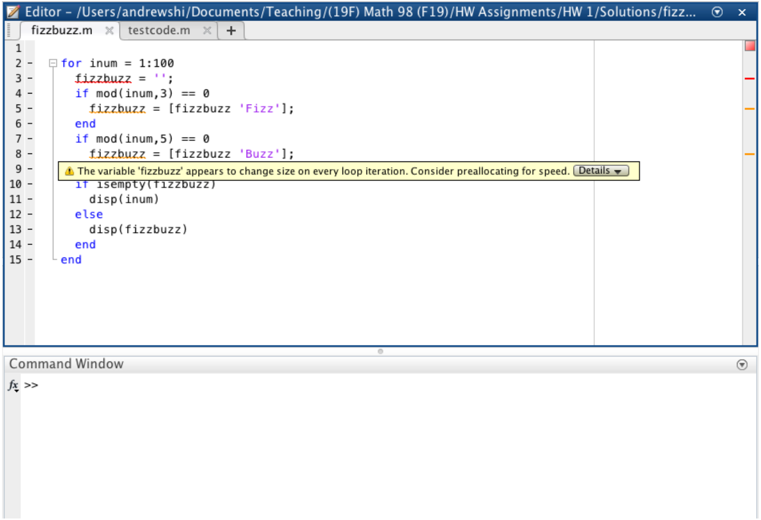
```
1
2 -   for inum = 1:100
3 -     fizzbuzz = '';
4 -     if mod(inum,3) == 0
5 -       fizzbuzz = [fizzbuzz 'Fizz'];
6 -     end
7 -     if mod(inum,5) == 0
8 -       fizzbuzz = [fizzbuzz 'Buzz'];
9 -     end
10 -    if isempty(fizzbuzz)
11 -      disp(inum)
12 -    else
13 -      disp(fizzbuzz)
14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

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

```
fx >>
```

Warnings

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



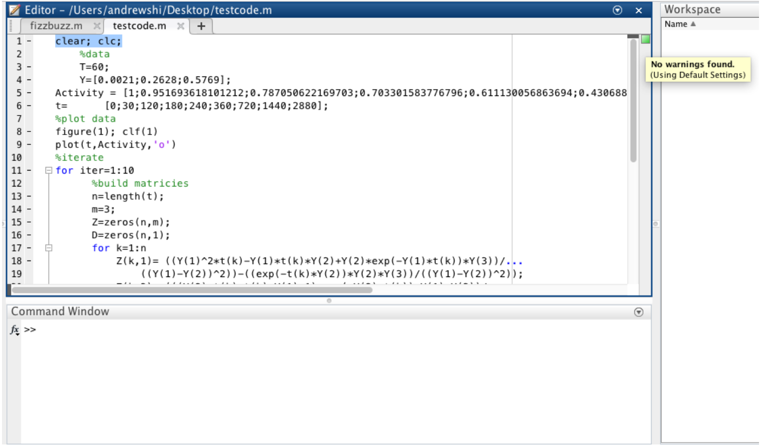
Editor – /Users/andrewshi/Documents/Teaching/(19F) Math 98 (F19)/HW Assignments/HW 1/Solutions/fizz...

```

for inum = 1:100
  fizzbuzz = '';
  if mod(inum,3) == 0
    fizzbuzz = [fizzbuzz 'Fizz'];
  end
  if mod(inum,5) == 0
    fizzbuzz = [fizzbuzz 'Buzz'];

  if isempty(fizzbuzz)
    disp(inum)
  else
    disp(fizzbuzz)
  end
end
```

⚠ The variable 'fizzbuzz' appears to change size on every loop iteration. Consider preallocating for speed.

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).



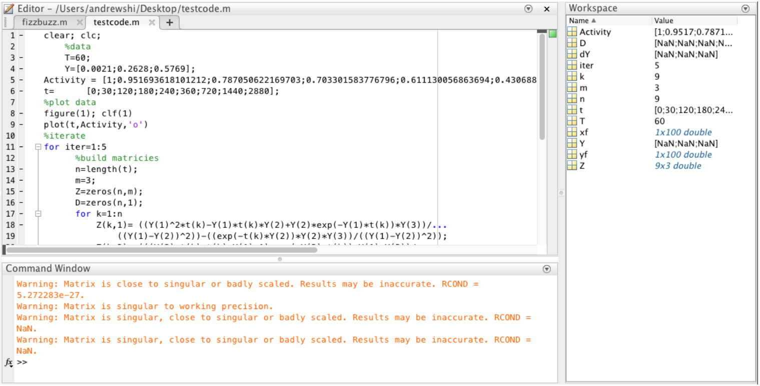
```
clear; clc;
    %data
    T=60;
    Y=[0.0021;0.2628;0.5769];
Activity = [1;0.951693618101212;0.787050622169703;0.703301583776796;0.611130056863694;0.430688
t=      [0;30;120;180;240;360;720;1440;2880];
%plot data
figure(1); clf(1)
plot(t,Activity,'o')
%iterate
for iter=1:10
      %build matricies
      n=length(t);
      m=3;
      Z=zeros(n,m);
      D=zeros(n,1);
      for k=1:n
          Z(k,1)= ((Y(1)^2*t(k)-Y(1)*t(k)*Y(2)+Y(2)*exp(-Y(1)*t(k))*Y(3))/...
              ((Y(1)-Y(2))^2))-((exp(-t(k)*Y(2))*Y(2)*Y(3))/((Y(1)-Y(2))^2));
```

No warnings found.
(Using Default Settings)

```
>>
```

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.



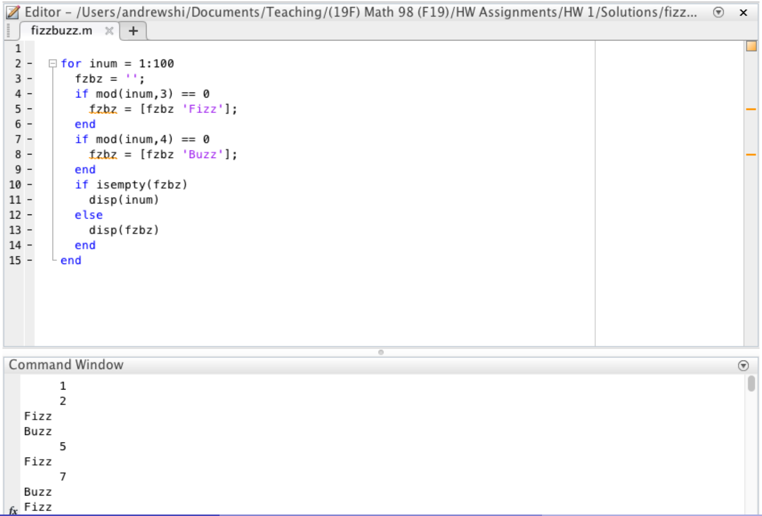
```
    clear; clc;
        %data
        T=60;
        Y=[0.0021;0.2628;0.5769];
    Activity = [1;0.951693618101212;0.787050622169703;0.703301583776796;0.611130056863694;0.430688
    t=      [0;30;120;180;240;360;720;1440;2880];
    %plot data
    figure(1); clf(1)
    plot(t,Activity,'o')
    %iterate
    for iter=1:5
          %build matricies
          n=length(t);
          m=3;
          Z=zeros(n,m);
          D=zeros(n,1);
          for k=1:n
              Z(k,1)= ((Y(1)^2*t(k)-Y(1)*t(k)*Y(2)+Y(2)*exp(-Y(1)*t(k))*Y(3))/...
                  ((Y(1)-Y(2))^2))-((exp(-t(k)*Y(2))*Y(2)*Y(3))/((Y(1)-Y(2))^2));
```

```
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.
fx >>
```

Name ▲	Value
Activity	[1;0.9517;0.7871...
D	[NaN;NaN;NaN;N...
dY	[NaN;NaN;NaN]
iter	5
k	9
m	3
n	9
t	[0;30;120;180;24...
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).

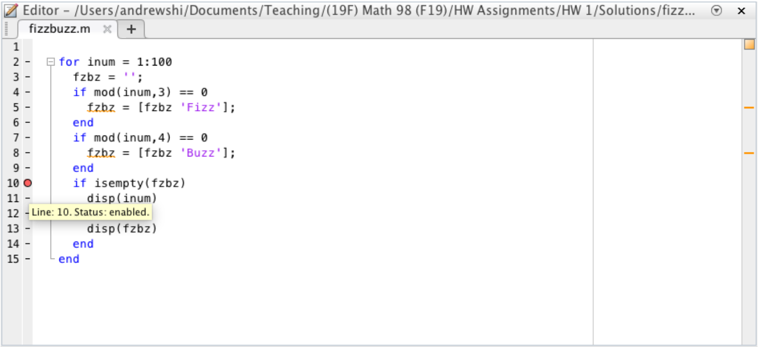

Editor – /Users/andrewshi/Documents/Teaching/(19F) Math 98 (F19)/HW Assignments/HW 1/Solutions/fizz...
fizzbuzz.m
+
Command Window
fx

```

for inum = 1:100
  fzbz = '';
  if mod(inum,3) == 0
    fzbz = [fzbz 'Fizz'];
  end
  if mod(inum,4) == 0
    fzbz = [fzbz 'Buzz'];
  end
  if isempty(fzbz)
    disp(inum)
  else
    disp(fzbz)
  end
end
```

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

Breakpoints

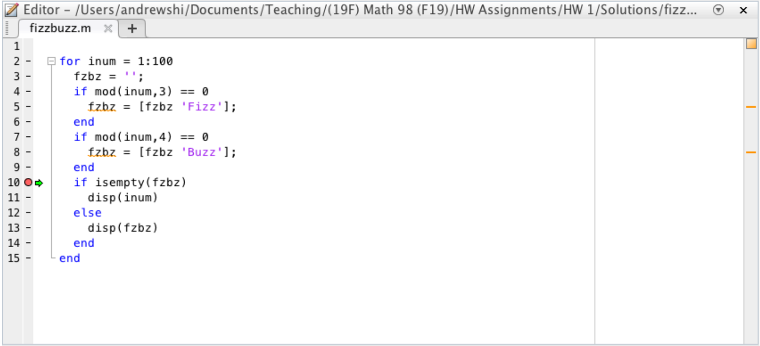
If I click one of the lines on the left it will create this red dot. This dot is called a breakpoint. I can easily remove it by clicking on it again.


Editor – /Users/andrewshi/Documents/Teaching/(19F) Math 98 (F19)/HW Assignments/HW 1/Solutions/fizz...
fizzbuzz.m
+
Line: 10. Status: enabled.

```

for inum = 1:100
  fzbz = '';
  if mod(inum,3) == 0
    fzbz = [fzbz 'Fizz'];
  end
  if mod(inum,4) == 0
    fzbz = [fzbz 'Buzz'];
  end
  if isempty(fzbz)
    disp(inum)

    disp(fzbz)
  end
end
```

Breakpoints

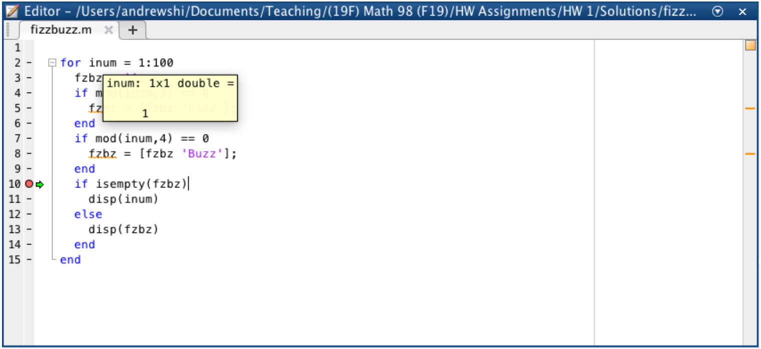
If I run my code again, it will stop the first time I hit the line with a breakpoint. The green arrow tells me where I am now.


Editor – /Users/andrewshi/Documents/Teaching/(19F) Math 98 (F19)/HW Assignments/HW 1/Solutions/fizz...
fizzbuzz.m
+

```

for inum = 1:100
  fzbz = '';
  if mod(inum,3) == 0
    fzbz = [fzbz 'Fizz'];
  end
  if mod(inum,4) == 0
    fzbz = [fzbz 'Buzz'];
  end
  if isempty(fzbz)
    disp(inum)
  else
    disp(fzbz)
  end
end
```

Breakpoints

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

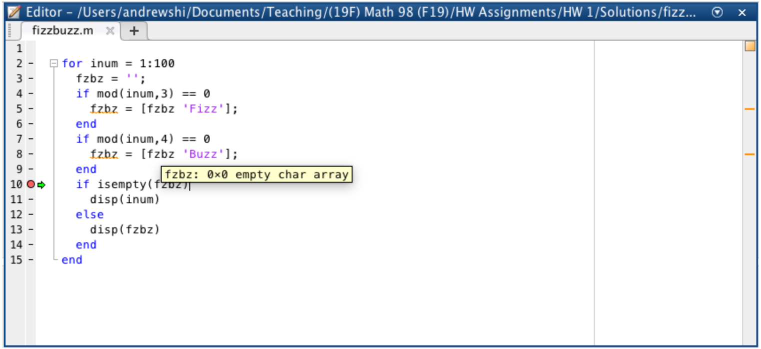

Editor – /Users/andrewshi/Documents/Teaching/(19F) Math 98 (F19)/HW Assignments/HW 1/Solutions/fizz...
fizzbuzz.m
+

```

for inum = 1:100
  fzbz = '';
  if mod(inum,3) == 0
    fzbz = [fzbz 'Fizz'];
  end
  if mod(inum,4) == 0
    fzbz = [fzbz 'Buzz'];
  end
  if isempty(fzbz)
    disp(inum)
  else
    disp(fzbz)
  end
end
```

```
inum: 1x1 double =

     1
```

Breakpoints

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

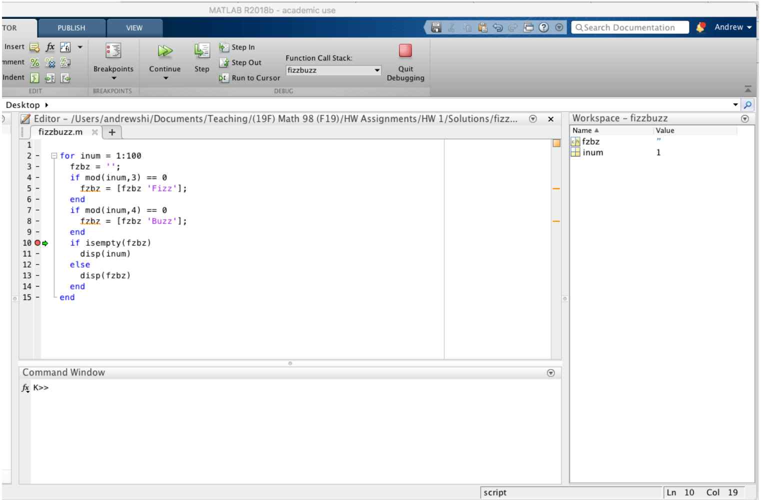

Editor – /Users/andrewshi/Documents/Teaching/(19F) Math 98 (F19)/HW Assignments/HW 1/Solutions/fizz...
fizzbuzz.m
+

```

for inum = 1:100
  fzbz = '';
  if mod(inum,3) == 0
    fzbz = [fzbz 'Fizz'];
  end
  if mod(inum,4) == 0
    fzbz = [fzbz 'Buzz'];
  end
  if isempty(fzbz)
    disp(inum)
  else
    disp(fzbz)
  end
end
```

fzbz: 0×0 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.



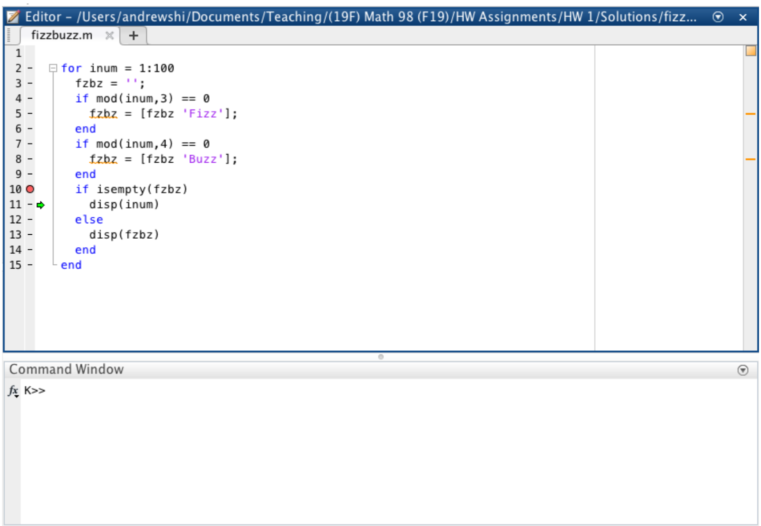
```
for inum = 1:100
  fzbz = '';
  if mod(inum,3) == 0
    fzbz = [fzbz 'Fizz'];
  end
  if mod(inum,4) == 0
    fzbz = [fzbz 'Buzz'];
  end
  if isempty(fzbz)
    disp(inum)
  else
    disp(fzbz)
  end
end
```

Name ▲	Value
fzbz	''
inum	1

```
K>>
```

Breakpoints

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

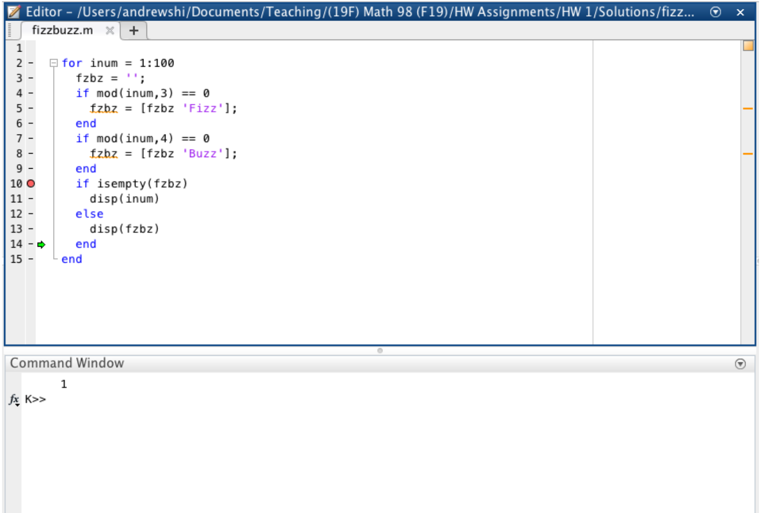

Editor – /Users/andrewshi/Documents/Teaching/(19F) Math 98 (F19)/HW Assignments/HW 1/Solutions/fizz...
fizzbuzz.m
+
Command Window

```

for inum = 1:100
  fzbz = '';
  if mod(inum,3) == 0
    fzbz = [fzbz 'Fizz'];
  end
  if mod(inum,4) == 0
    fzbz = [fzbz 'Buzz'];
  end
  if isempty(fzbz)
    disp(inum)
  else
    disp(fzbz)
  end
end
```

```
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?)

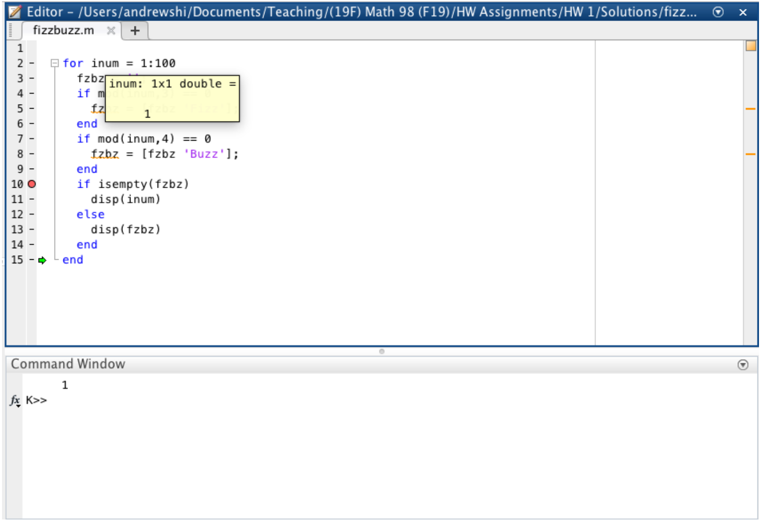

Editor – /Users/andrewshi/Documents/Teaching/(19F) Math 98 (F19)/HW Assignments/HW 1/Solutions/fizz...
fizzbuzz.m
+
Command Window

```
 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
```

```
      1
fx K>>
```

Breakpoints

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


Editor – /Users/andrewshi/Documents/Teaching/(19F) Math 98 (F19)/HW Assignments/HW 1/Solutions/fizz...
fizzbuzz.m
+
Command Window

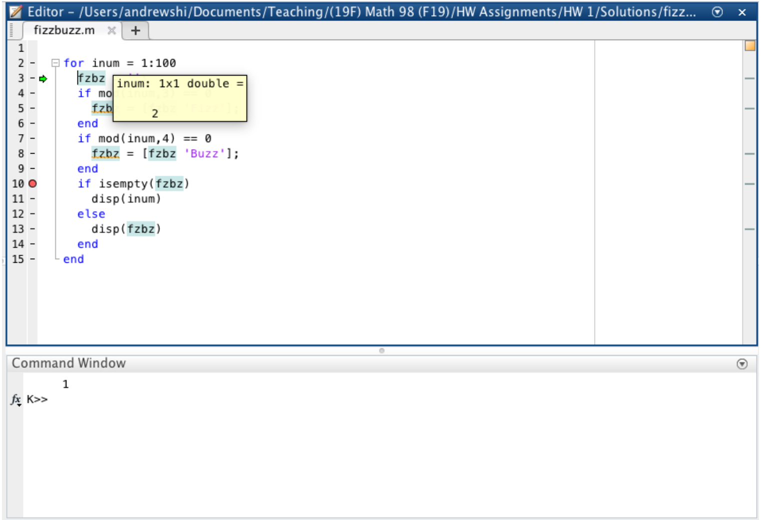
```

for inum = 1:100
    fzbz
    if mod(inum,3) == 0
        fzbz = [fzbz 'Fizz'];
    end
    if mod(inum,4) == 0
        fzbz = [fzbz 'Buzz'];
    end
    if isempty(fzbz)
        disp(inum)
    else
        disp(fzbz)
    end
end
```

```
inum: 1x1 double =

     1
```

```
     1
fx K>>
```

Breakpoints

And now `inum = 2`.



Editor – /Users/andrewshi/Documents/Teaching/(19F) Math 98 (F19)/HW Assignments/HW 1/Solutions/fizz...

fizzbuzz.m

```matlab

for inum = 1:100
  fzbz = '';
  if mod(inum,3) == 0
    fzbz = [fzbz 'Fizz'];
  end
  if mod(inum,4) == 0
    fzbz = [fzbz 'Buzz'];
  end
  if isempty(fzbz)
    disp(inum)
  else
    disp(fzbz)
  end
end
```

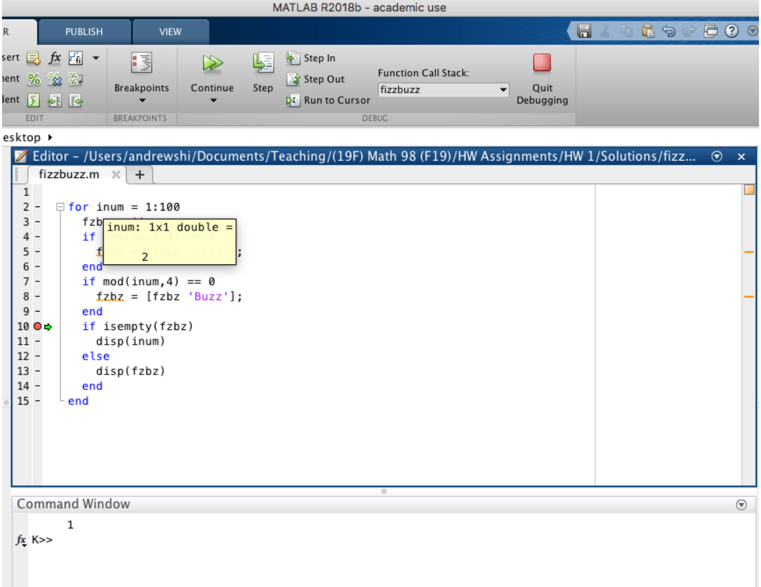
```
inum: 1x1 double =

     2
```

Command Window

```
     1
K>>
```

Breakpoints

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



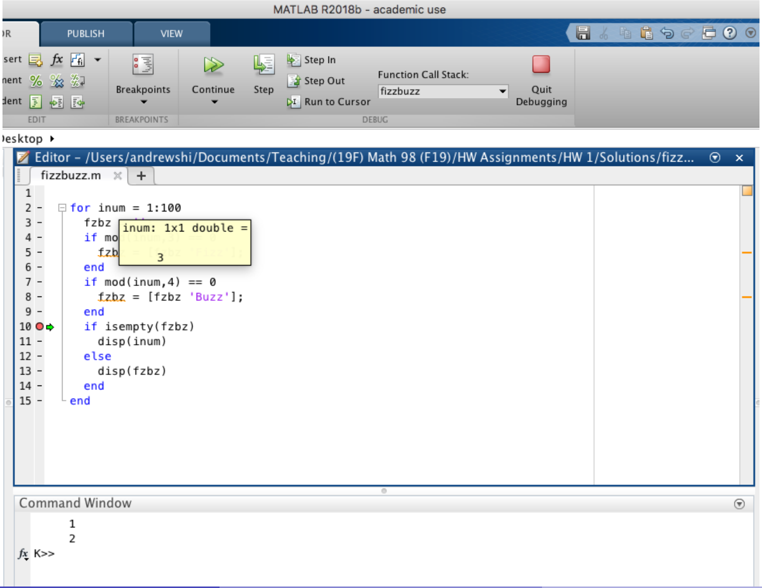
```

for inum = 1:100
  fzb
  if
    f                           ;
  end
  if mod(inum,4) == 0
    fzbz = [fzbz 'Buzz'];
  end
  if isempty(fzbz)
    disp(inum)
  else
    disp(fzbz)
  end
end
```

```
inum: 1x1 double =

    2
```

```
     1
K>>
```

Breakpoints

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



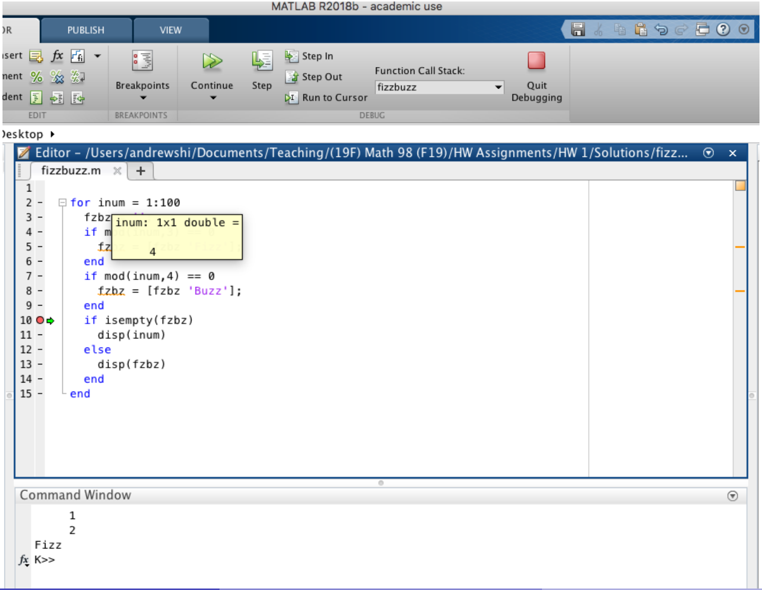
```

for inum = 1:100
  fzbz
  if mod(inum,3) == 0
    fzbz = [fzbz 'Fizz'];
  end
  if mod(inum,4) == 0
    fzbz = [fzbz 'Buzz'];
  end
  if isempty(fzbz)
    disp(inum)
  else
    disp(fzbz)
  end
end
```

```
inum: 1x1 double =

     3
```

```
     1
     2
K>>
```

Breakpoints

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



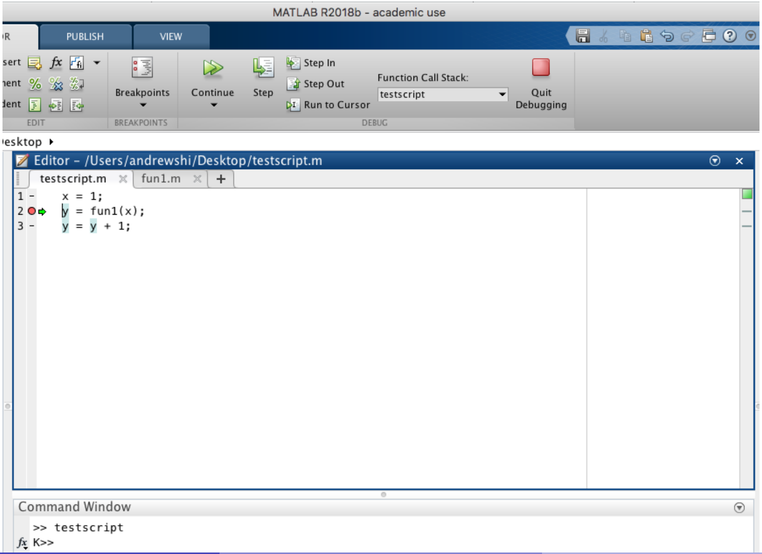
```
for inum = 1:100
    fzbz = '';
    if mod(inum,3) == 0
      fzbz = [fzbz 'Fizz'];
    end
    if mod(inum,4) == 0
      fzbz = [fzbz 'Buzz'];
    end
    if isempty(fzbz)
      disp(inum)
    else
      disp(fzbz)
    end
end
```

```
inum: 1x1 double =

     4
```

```
     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.

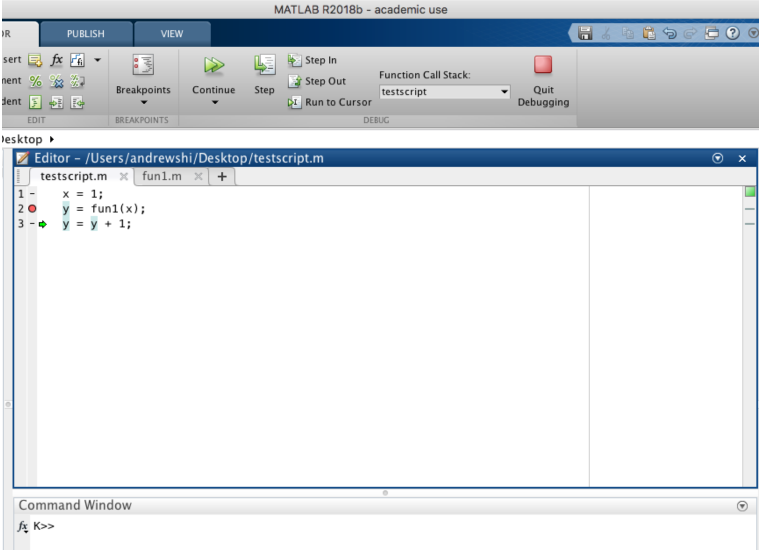


```
x = 1;
y = fun1(x);
y = y + 1;
```

```
>> testscript
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.



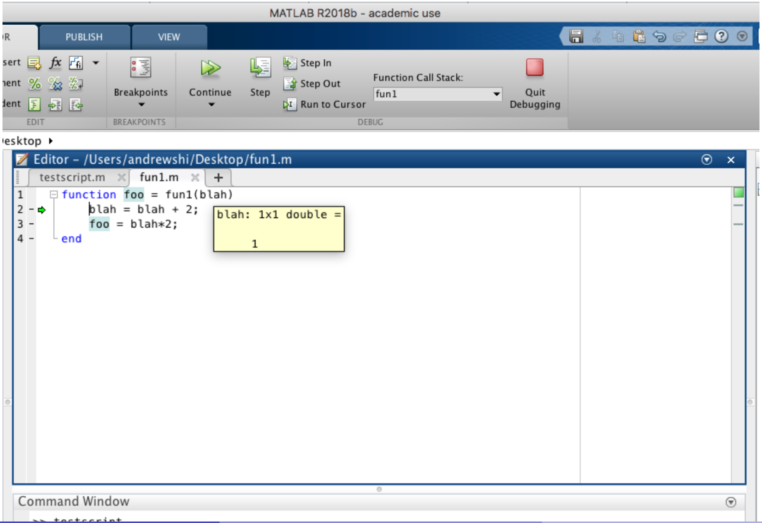
MATLAB R2018b - academic use

```
x = 1;
y = fun1(x);
y = y + 1;
```

```
K>>
```

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.



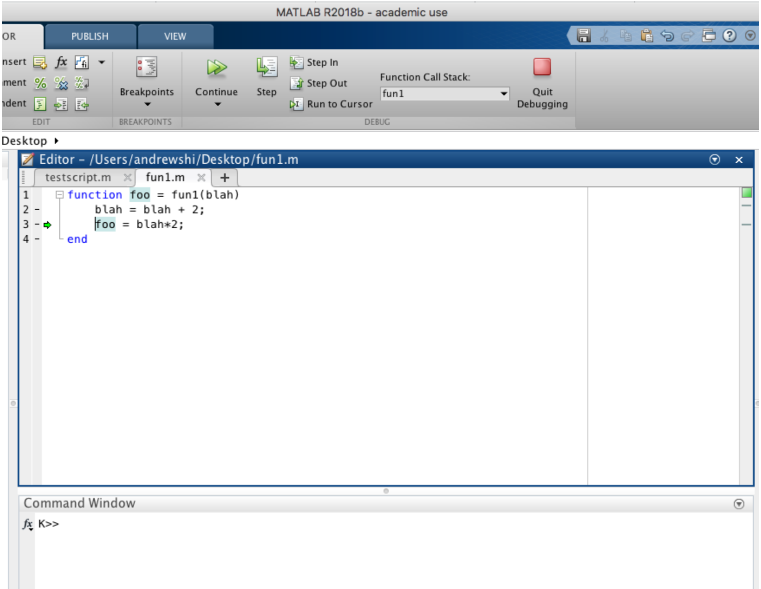
```
function foo = fun1(blah)
    blah = blah + 2;
    foo = blah*2;
end
```

```
blah: 1x1 double =

     1
```

```
>> testscript
```

Step In/Out

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

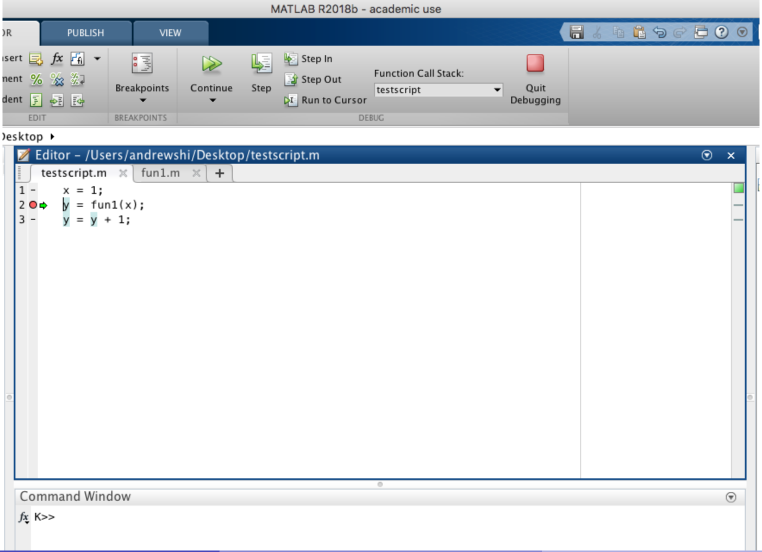


```
function foo = fun1(blah)
    blah = blah + 2;
    foo = blah*2;
end
```

```
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.

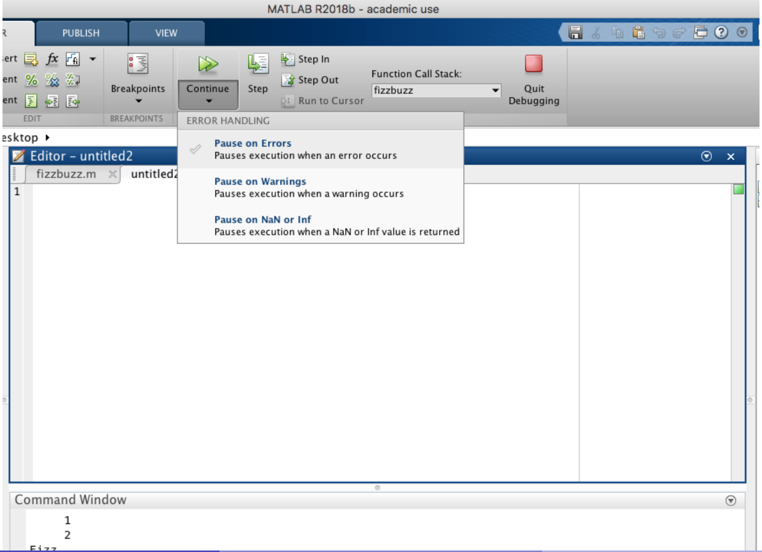


```
x = 1;
y = fun1(x);
y = y + 1;
```

```
K>>
```

Run Options

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



ERROR HANDLING

✓ **Pause on Errors**
Pauses execution when an error occurs

Pause on Warnings
Pauses execution when a warning occurs

Pause on NaN or Inf
Pauses execution when a NaN or Inf value is returned

```
1
2
Fizz
```

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:

```
[max] = 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.