Week 6 Tutorial 3

The purpose of this tutorial is to demonstrate a common error in MATLAB that can be difficult to identify; saving a MATLAB program with an invalid name

```
% Always clear workspace variables before a new tutorial or program.
clear
clc
```

Edit the code below and update the variable named **name** with your name for this tutorial in the code below.

```
name="";
fprintf("Output for Tutorial_06_3 run by %s.\n", name)
```

Error - Overriding constants in MATLAB

There are hundreds of constants in MATLAB, even more when you start adding add-ons. It is difficult to know if your variable name "hides" an existing constant. Follow along in the example and see how this can negatively affect your program.

Matlab allows you to use MATLAB function names and constants as variables. If you do this your variable will take precedence over the built-in constant name in the workspace. Here we'll print the predefined values of a few constants.

```
% Print the predefined values
fprintf('Default Values of Matlab Constants:\n\n')
fprintf('pi = %g , eps = %g \nrealmin = % g, realmax = %g\n\n', ...
    pi,eps,realmin,realmax)
```

Override the constants

Now, create your own variables with the names **pi**, **eps**, **realmin**, **realmax** and set them to some arbitrary value.

```
% Override built-in variables by assinging your own with the same names


% MATLAB allows these constant names to be used without giving a warning
fprintf('Matlab Constants set to 0:\n\n')
fprintf('pi = %g , eps = %g \nrealmin = % g, realmax = %g\n\n', ...
        pi,eps,realmin,realmax)
```

You can see, the values for these constants now contain the values from your workspace variables as your workspace takes precedence over any built-in constants.

Clear variables to restore built-in constants

We can easily reset the constants by clearing out the variable names.

```
% To restore the default values of the MATLAB constants you need to clear
% the variable names
clear  % clears all variable names in the workspace
fprintf('Matlab constants after the clear command:\n\n')
fprintf('pi = %g , eps = %g \nrealmin = % g, realmax = %g\n\n', ...
        pi,eps,realmin,realmax)
```

Error - Overriding functions in MATLAB

Regarding functions, there are likely hundreds, if not thousands of built-in functions in MATLAB. As with constants, there are even more when you start adding add-ons. Follow along in the example and see how this can negatively affect your program.

As you'll see in the following, you can override or "hide" a function simply by creating a variable of the same name, the same works if you were to create a function that shares a name with a built-in variable or function name.

```
% Using the functions sum() and prod() prior to overriding
x = 1:5;
fprintf('Matlab Functions Prior to Overriding:\n\n')
fprintf('x = %i  %i  %i  %i  %i\n',x)
fprintf('sum(x) = %2i , prod(x) = %g\n\n',sum(x), prod(x))
```

Override the functions

Now, override or "mask" the built-in sum and prod functions by creating variables of the same name.

```
% Create variables for sum and prod, assign some arbitrary value


% Using the variable names sum and prod, Matlab does not warn you that you
% are overriding the functions.
fprintf('Overriding function Names:\n\n')
fprintf('sum = %2i , prod = %g\n\n',sum, prod)

% Using the functions sum() and prod() after overriding will cause an error
% Note that when you comment out the clear sum prod line you will get
% the error message:

% ??? Index exceeds matrix dimensions.
% Error in ==> Tutorial_09_1 at 91.

% This error appears because Matlab interprets sum(x) and prod(x) as your
% attempt to access elements in vectors or matrices that do not exist.
fprintf('Matlab Functions after Overriding:\n\n')
fprintf('x = %i  %i  %i  %i  %i\n',x)
% To create the error comment the next line (add a % a the beginning)
fprintf('sum(x) = %2i , prod(x) = %g\n\n',sum(x), prod(x))
```

You should notice, when running this section, or the whole script, you will receive an error as follows:

"Index exceeds the number of array elements. Index must not exceed 1."

This may be confusing at first but, let's think about this for a second. A function nearly always uses open and close parenthesis. Subscripting a matrix is technically a function and, as you see, it shares the same format as a function. For example `sum(x)` will sum the value(s) in x. However, if `sum` is a variable `sum(x)` will access the nth element in the `sum` matrix.

Now, you likely created a scalar variable so you get this error of exceeding array elements since you're trying to access the elements in the positions that match the value of x.

In the worst-case scenario, you created a matrix of more than 5 elements, now if you try to run `sum(x)` you just simply get a vector or, if x is a scalar you just get a single value. Either of which will be very confusing as MATLAB will not give you an error, instead you will get a valid output that might appear to be the result of `sum(x)` when it is merely accessing the values of `sum`.

Clear the sum and prod variables and, for good measure, comment out the last line in the previous code block to rid ourselves of this trap.

```
% Clear the variables sum and prod

```

Avoid overriding or hiding variables and functions

It is best to avoid using variable names that match those of MATLAB functions and constants. To check to see if a name is used for a MATLAB function or constant type help followed by the variable name in the command window or your script file. If MATLAB uses that name a description will appear. If the name is not used the message "**name** not found." will be displayed.

```
help eps  % matlab will display a description of eps
help prod % matlab will display a description of the prod() function
help student  % not found, you may use this variable in your program
```

It can also be helpful to think about what type of manipulations you are running in your program. If you are going to need the sum, prod, etc, functions then definitely avoid creating variables of the same name.

Example Output:

Run this tutorial from the **Command Window** and ensure your output matches the following.

```
Output for Tutorial_06_3 run by Geoff Berl.
Default Values of Matlab Constants:

pi = 3.14159 , eps = 2.22045e-16
realmin =  2.22507e-308, realmax = 1.79769e+308

Matlab Constants set to 0:

pi = 0 , eps = 0
realmin =  0, realmax = 0

Matlab constants after the clear command:

pi = 3.14159 , eps = 2.22045e-16
realmin =  2.22507e-308, realmax = 1.79769e+308

Matlab Functions Prior to Overriding:

x = 1  2  3  4  5
sum(x) = 15 , prod(x) = 120

Overriding function Names:

sum =  0 , prod = 0

Matlab Functions after Overriding:

x = 1  2  3  4  5
```

eps Spacing of floating point numbers.
eps(X) is the positive distance from ABS(X) to the next larger in magnitude floating point number of the same precision as X. X may be either double precision or single precision. For all X, **eps**(X) is equal to **eps**(ABS(X)).

eps, with no arguments, is the distance from 1.0 to the next larger double precision number, that is **eps** with no arguments returns 2^(-52).

eps(CLASSNAME) returns the positive distance from 1.0 to the next floating point number in the precision of CLASSNAME.

eps('double') is the same as **eps**, or **eps**(1.0).
eps('single') is the same as **eps**(single(1.0)), or single(2^-23).

eps('like', Y) returns the positive distance from 1.0 to the next floating point number in the precision of the numeric variable Y, with the same data type, sparsity, and complexity (real or complex) as the numeric variable Y.

Except for numbers whose absolute value is smaller than REALMIN, if 2^E <= ABS(X) < 2^(E+1), then

```
   eps(X) returns 2^(E-23) if ISA(X,'single')
   eps(X) returns 2^(E-52) if ISA(X,'double')
```

For all X of class double such that ABS(X) <= REALMIN, **eps**(X) returns 2^(-1074). Similarly, for all X of class single such that ABS(X) <= REALMIN('single'), **eps**(X) returns 2^(-149).

Replace expressions of the form

```
   if Y < eps * ABS(X)
```

with

```
   if Y < eps(X)
```

Example return values from calling **eps** with various inputs are presented in the table below:

Expression	Return Value
eps(1/2)	2^(-53)
eps(1)	2^(-52)
eps(2)	2^(-51)
eps(realmax)	2^971
eps(0)	2^(-1074)
eps(realmin/2)	2^(-1074)
eps(realmin/16)	2^(-1074)
eps(Inf)	NaN
eps(NaN)	NaN
eps(single(1/2))	2^(-24)
eps(single(1))	2^(-23)
eps(single(2))	2^(-22)
eps(realmax('single'))	2^104
eps(single(0))	2^(-149)
eps(realmin('single')/2)	2^(-149)
eps(realmin('single')/16)	2^(-149)
eps(single(Inf))	single(NaN)
eps(single(NaN))	single(NaN)

See also realmax, realmin.

```
prod Product of elements.
   P = prod(X) is the product of the elements of the vector X. If X is a
   matrix, P is a row vector with the product over each column. For
   N-D arrays, prod(X) operates on the first non-singleton dimension.

   prod(X,'all') computes the product of all elements of X.

   prod(X,DIM) operates along the dimension DIM.

   prod(X,VECDIM) operates on the dimensions specified in the vector
   VECDIM. For example, prod(X,[1 2]) operates on the elements contained
   in the first and second dimensions of X.

   prod(...,OUTTYPE) specifies the type in which the product is
   performed, and the type of P. Available options are:

   'double'    -  P has class double for any input X
   'native'    -  P has the same class as X
   'default'   -  If X is floating point, that is double or single,
                  P has the same class as X. If X is not floating point,
                  P has class double.

   prod(...,NANFLAG) specifies how NaN (Not-A-Number) values are
   treated. The default is 'includenan':

   'includenan' - the product of a vector containing NaN values is also NaN.
   'omitnan'    - the product of a vector containing NaN values
                  is the product of all its non-NaN elements. If all
                  elements are NaN, the result is 1.

   Examples:
       X = [0 1 2; 3 4 5]
       prod(X,1)
       prod(X,2)

       X = int8([5 5 5 5])
       prod(X)                % returns double(625), accumulates in double
       prod(X,'native')       % returns int8(127), because it accumulates in
                              % int8, but overflows and saturates.

   See also sum, cumprod, diff.

   Documentation for prod
   Other uses of prod
```

```
--- student not found. Showing help for isstudent instead. ---

 isstudent True for MATLAB Student Version.
    isstudent returns true for the Student Version of MATLAB
    and false for the commercial version.

    See also ver, version, license, ispc, isunix, desktop.

    Documentation for isstudent
```