Naming Variables

MATLAB variable names must begin with a letter, which may be followed by any combination of letters, digits, and underscores. MATLAB distinguishes between uppercase and lowercase characters, so `A` and `a` are not the same variable.

Although variable names can be of any length, MATLAB uses only the first `N` characters of the name, (where `N` is the number returned by the function `namelengthmax`), and ignores the rest. Hence, it is important to make each variable name unique in the first `N` characters to enable MATLAB to distinguish variables.

```
N = namelengthmax
N =
    63
```

The `genvarname` function can be useful in creating variable names that are both valid and unique. See the `genvarname` reference page to find out how to do this.

Verifying a Variable Name

You can use the `isvarname` function to make sure a name is valid before you use it. `isvarname` returns 1 if the name is valid, and 0 otherwise.

```
isvarname 8th_column
ans =
     0                    % Not valid - begins with a number
```

Avoid Using Function Names for Variables

When naming a variable, make sure you are not using a name that is already used as a function name, either one of your own M-file functions or one of the functions in the MATLAB language. If you define a variable with a function name, you will not be able to call that function until you either remove the variable from memory with the `clear` function, or invoke the function using `builtin`.

For example, if you enter the following command, you will not be able to use the MATLAB `disp` function until you clear the variable with `clear disp`.

```
disp = 50;
```

To test whether a proposed variable name is already used as a function name, use

```
which -all variable_name
```

Potential Conflict with Function Names

There are some MATLAB functions that have names that are commonly used as variable names in programming code. A few examples of such functions are `i`, `j`, `mode`, `char`, `size`, and `path`.

If you see a need to use a particular function name such as one of these for a variable, and you determine that you have no need to call that function in your program, you should be aware that there is still a possibility for conflict. See the

following two examples:

- Variables Loaded From a MAT−File
- Variables In Evaluation Statements

Variables Loaded From a MAT−File. The function shown below loads previously saved data from MAT−file `settings.mat` and is supposed to display the value of one of the loaded variables, `mode`. However, `mode` is also the name of a MATLAB function and, in this case, MATLAB interprets it as the function and not the variable loaded from the MAT−file:

```
function show_mode
load settings;
whos mode
fprintf('Mode is set to %s\n', mode)
```

Assume that `mode` already exists in the MAT−file. Execution of the function shows that, even though `mode` is successfully loaded into the function workspace as a variable, when MATLAB attempts to operate on it in the last line, it interprets `mode` as a function. This results in an error:

```
show_mode
  Name        Size                     Bytes  Class

  mode        1x6                         12  char array

Grand total is 6 elements using 12 bytes

??? Error using ==> mode
Not enough input arguments.

Error in ==> show_mode at 4
fprintf('Mode is set to %s\n', mode)
```

Because MATLAB parses function M−files before they are run, it needs to determine before runtime which identifiers in the code are variables and which are functions. The function in this example does not establish `mode` as a variable name and, as a result, MATLAB interprets it as a function name instead.

There are at least two ways to make this function work as intended without having to change the variable name. Both indicate to MATLAB that the name represents a variable, and not a function:

- Name the variable explicitly in the `load` statement:

  ```
  function show_mode
  load settings mode;
  whos mode
  fprintf('Mode is set to %s\n', mode)
  ```

- Initialize the variable (e.g., set it to an empty matrix or empty string) at the start of the function:

  ```
  function show_mode
  mode = '';
  load settings;
  whos mode
  fprintf('Mode is set to %s\n', mode)
  ```

Variables In Evaluation Statements. Variables used in evaluation statements such as `eval`, `evalc`, and `evalin` can also be mistaken for function names. The following M−file defines a variable named `length` that conflicts with MATLAB `length` function:

```
function find_area
eval('length = 12;  width = 36;');
fprintf('The area is %d\n', length .* width)
```

The second line of this code would seem to establish `length` as a variable name that would be valid when used in the statement on the third line. However, when MATLAB parses this line, it does not consider the *contents* of the string that is to be evaluated. As a result, MATLAB has no way of knowing that `length` was meant to be used as a variable name in this program, and the name defaults to a function name instead, yielding the following error:

```
find_area
??? Error using ==> length
Not enough input arguments.
```

To force MATLAB to interpret `length` as a variable name, use it in an explicit assignment statement first:

```
function find_area
length = [];
eval('length = 12;  width = 36;');
fprintf('The area is %d\n', length .* width)
```