MATLAB: Frequently made mistakes

Preamble: 95% of coding mistakes by beginners fall into of one of these 10 categories. Generally speaking, try to welcome the red ink, as frustrating as it might be at first. MATLAB is telling you that it didn't understand one of your commands. The worst kind of error is the one where is not doing what you think it is doing, but you are never alerted to this fact because MATLAB thinks it understands what you want. This can be quite dangerous. That said, MATLAB's error messages can be cryptic, so we are decoding the ones you are most likely to encounter.

1. **Not closing a parenthesis**: Every parenthesis that is opened must be closed again. Importantly, the kinds of parenthesis must match – MATLAB distinguishes 3 kinds: Regular parentheses ( ), square brackets [ ], and curly braces { }. They all have different purposes.

Example of a typical error and error message that you might see (note that MATLAB now makes a guess as what you might have meant):

```
>> A(3
 A(3
    |
Error: Expression or statement is incorrect--possibly unbalanced (, {, or [.

Did you mean:
>> A(3)
```

But don't add more than you need (they need to match exactly - not too many, not too few):

```
>> size(A))
 size(A))
         |
Error: Unbalanced or unexpected parenthesis or bracket.
```

2. **Trying to create an illegal variable name**: MATLAB is quite permissive when it comes to acceptable variable names, but there are 3 exceptions: They can't start with a number, there can be no blank space in the middle (it has to be one word) and they can't contain special characters.

The kind of error message you will get depends on which of these rules you violate. In turn:
a) Starting a variable name with a number, MATLAB politely tells you that it literally does not know what you are trying to say. What you said was "unexpected".

```
>> 5times = 120
 5times = 120
  |
Error: Unexpected MATLAB expression.
```

b) Having a space within a variable name. It has to be one word. Cryptic error message:

```
>> times five = 120
Error using .*
Too many input arguments.
```

c) Having a special character in a variable name. Can't be done:

```
>> times* = 5
 times* = 5
        |
Error: The expression to the left of the equals sign is not a valid
target for an assignment.
```

3. **Trying to access matrix elements with invalid indices:** MATLAB indexes from 1 (the first element is element 1), so indices can't be 0 or negative. And they have to be integers. Example:

```
>> A(0)
Subscript indices must either be real positive integers or logicals.
>> A(1.5)
Subscript indices must either be real positive integers or logicals.
```

4. **Trying to access matrix elements that exceed the size of the matrix:**

```
>> A = [1 2 3];
>> A(4)
Index exceeds matrix dimensions.
```

5. **Trying to access a variable that doesn't exist:** This usually happens when trying to access a variable that no longer exists (has been cleared from memory), hasn't been declared yet (Matlab reads and executes lines in strictly chronological fashion) or simply when misspelling a variable that has been declared (although Matlab now ventures a guess in this case):

```
>> times5 = 5;
>> tines5
Undefined function or variable 'tines5'.

Did you mean:
>> times5
```

6. **Using a variable name that corresponds to a function, then trying to use that function:** Generally speaking, it is advisable to use descriptive variable names ("participantPoolSize" instead of "size" to avoid collisions with functions, as they are usually short). If you want to be sure whether the name of a variable you plan to use corresponds to that of an existing function you can type *which* followed by the variable name. If nothing comes up, you are good to go. What makes this error particularly tricky is that the error message you will see depends on the particular function you replaced. The dead giveaway that you made this error is simply that a function that you used many times before is suddenly no longer working. Note that the last of these examples is the most insidious, as it doesn't even throw an error message – it dutifully does what you asked it to do (but probably not what you meant), namely to access the first element (as denoted by the content of A) of "mean" – a classic man-machine misunderstanding.

```
>> size = 7;
>> size(A)
Index exceeds matrix dimensions.

>> sin = 180;
>> sin(pi)
Subscript indices must either be real positive integers or logicals.

>> mean = 5;
>> A = 1;
>> mean(A)

ans =
     5
```

7. **There can be no holes in a matrix:** Each row and column in a matrix needs to have the same number of elements. If this isn't so, Matlab will throw an error:

```
> A = [1 2 3;4 5]
Error using vertcat
Dimensions of matrices being concatenated are not consistent.
```

8. **When adding matrices, make sure that they have the same dimensionality**:

```
>> A = [1 2;3 4];
>> B = [1 1 1; 2 2 2];
>> C = A + B
Error using  +
Matrix dimensions must agree.
```

This might seem like a subtle point, but this also applies to adding *to* a matrix. In the commands below, assigning A to the first column of C sets C up as a 5x1 matrix, but B is only 4 elements long. You can't then assign B to the second column of C. The two assignment dimensions don't match, and Matlab will – helpfully if frustratingly – tell you so.

```
>> A = [1 2 3 4 5];
>> B = [6 7 8 9];
>> C(1,:) = A; C(2,:) = B;
Subscripted assignment dimension mismatch.
```

9. **When plotting vectors, they must be the same length:** Generally speaking, every function that does something with pairs of vectors, the number of elements in the two vectors needs to match:

```
>> A = [1 2 3 4 5];
>> B = [6 7 8 9];
>> plot(A,B)
Error using plot
Vectors must be the same length.
```

10. **When specifying attribute-value pairs, make sure to give it what it expects:** A number if it expects a number and a string or character when a string or character is expected. Characters or strings are denoted by single scare quotes, whereas numbers lack quotation marks. All attributes of a plotted line can be specified, for instance "linewidth", which expects a number or "marker", which expects a string:

```
>> plot(A,B,'linewidth','3')
Error using plot
Value not a numeric scalar
```

```
>> plot(A,B,'marker',square)
Undefined function or variable 'square'.
```

The same goes for parsing errors (everything inside '' is seen as part of the attribute (and colored purple – that is the dead giveaway):

```
>> plot(A,B,'linewidth,'3)
 plot(A,B,'linewidth,'3)
                      |
Error: Unexpected MATLAB expression.
```

Finally, you can see from all of this that MATLAB takes you very literally. It will not "catch your drift". This is a feature, not a bug. It yields consistency. What an error messages indicates is that MATLAB does not understand your command well enough to execute it, but is aware of this fact. Again, this is a good thing. The worst mistake is the kind that is serious but goes unnoticed. And remember: The computer takes you 100% literally. It has to, because you have no common ground and it is an advantage because behavior is consistent.