GEOS F436/636 Beyond the Mouse

Christine (Chris) Waigl
University of Alaska Fairbanks – Fall 2018

Week 11: Debugging strategies

FIRST: Voting is important.

- If you are registered anywhere in Alaska and haven't voted yet, it's more important to get yourself to the Wood Center now.
- If you don't know what's on the ballot: https://www.ballotready.org/
- If you don't think you're registered, but have applied for the PFD, you probably are registered. Alaska has automatic voter registration.

Coding errors are a fact of life

During this class or your own work, what experiences with coding errors have you made? What sources of errors did you experience? What strategies have you applied to correct your code?

Are there any specific techniques or topics you are curious about?

source: internet meme, origin unknown

Topics for week 11

- Errors and code: types of errors
- Understanding error messages
- The basics of testing
- Using a debugger
- Avoiding errors that stop your program: try-catch mechanism

ALSO: I will send out a note with requirements for your project this week. You can get started with creating your project repository in GitHub!

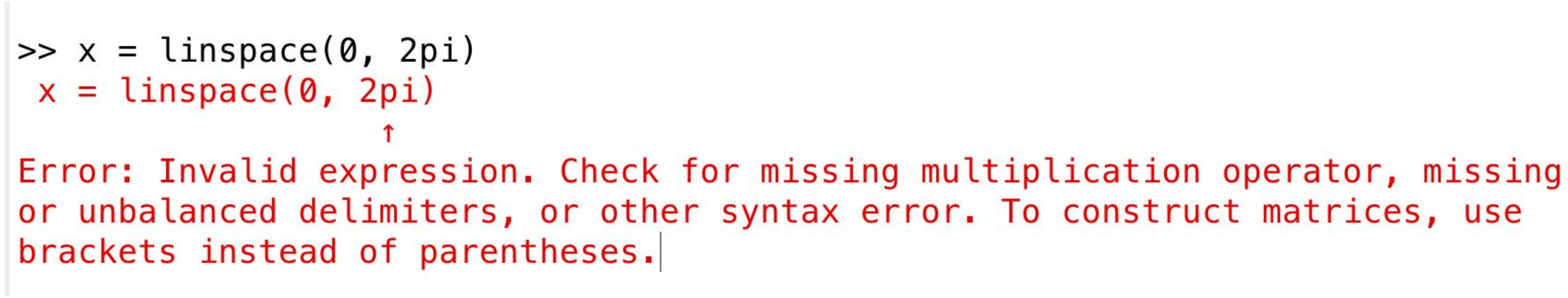
Types of programming errors

- **syntax errors**: you violate the rules of the language and your interpreter or compiler can't make sense of your code.
- **runtime errors**: The syntax is ok, but when you run the program, execution stops because something is wrong.
- **logical errors**: The syntax is ok and the program finishes, but the result is wrong.
- hidden or latent errors: The syntax is ok, the program finishes typically, and when it does the result is most of the time correct, but sometimes either your program stops or the result is incorrect.

Can you think of any examples?

Some errors (though by far not all!) come with error messages. You should always read them carefully.

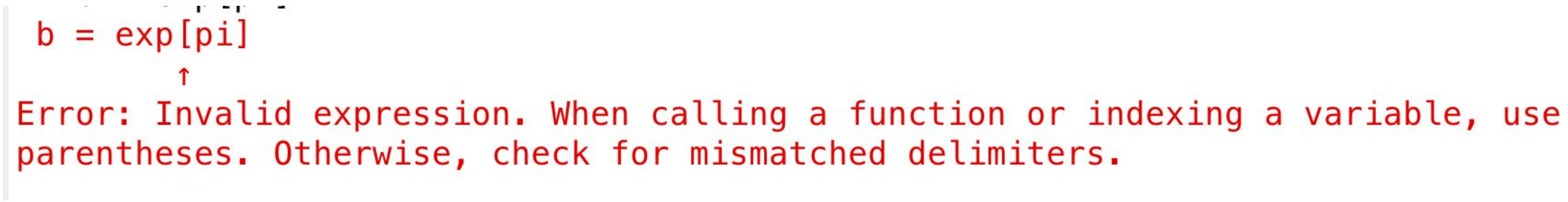
What kind of error is this? What's wrong?



```
>> x = linspace(0, 2pi)
 x = linspace(0, 2pi)
                  ↑
Error: Invalid expression. Check for missing multiplication operator, missing
or unbalanced delimiters, or other syntax error. To construct matrices, use
brackets instead of parentheses.
```

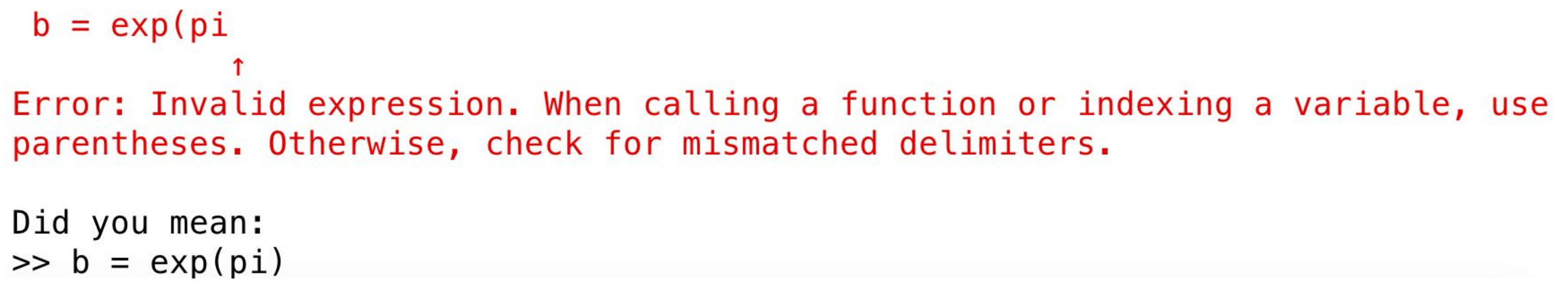
A syntax error. We forgot the operator * to multiply 2 and pi.

Kind of error? What is wrong?



```
 b = exp[pi]
        ↑
Error: Invalid expression. When calling a function or indexing a variable, use
parentheses. Otherwise, check for mismatched delimiters.
```

Call function using parentheses (), not brackets [] or braces {}



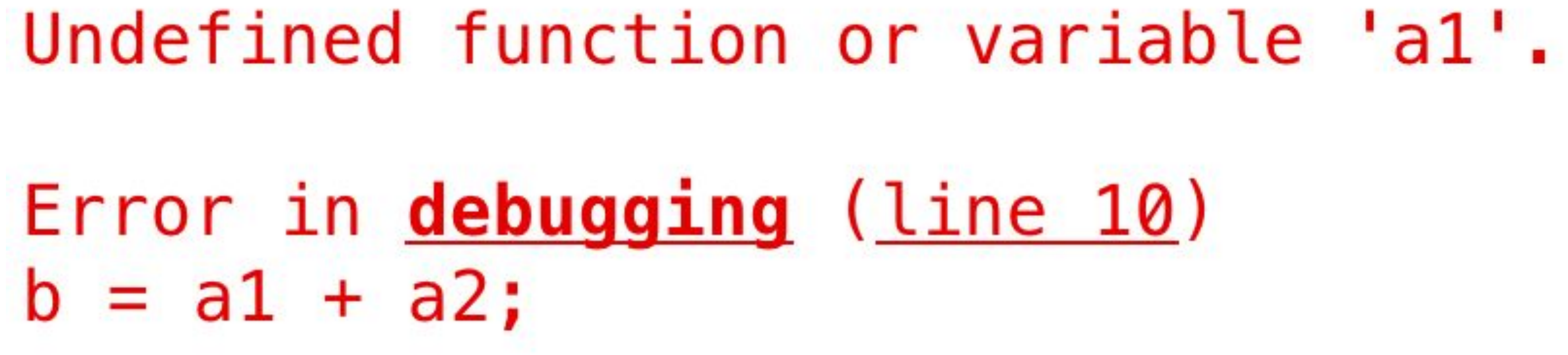
```
 b = exp(pi
           ↑
Error: Invalid expression. When calling a function or indexing a variable, use
parentheses. Otherwise, check for mismatched delimiters.

Did you mean:
>> b = exp(pi)
```

MATLAB syntax errors

- Come with the error message "invalid expression"
- Typically: unmatched parentheses, missing operators...

Kind of error? What is wrong?

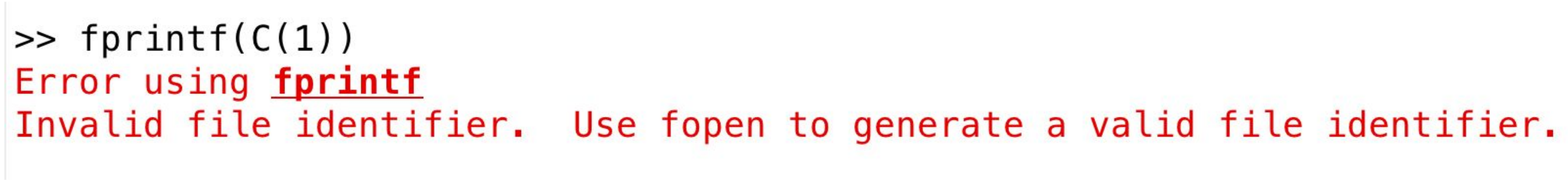


```
Undefined function or variable 'a1'.

Error in debugging (line 10)
b = a1 + a2;
```

A runtime error. The variable a1 was not defined.

Kind of error? What is wrong?



```
>> fprintf(C(1))
Error using fprintf
Invalid file identifier.  Use fopen to generate a valid file identifier.
```

Another runtime error. Not clear what it means! So we have to look more closely .... see live session.

NOTE: Runtime errors in MATLAB usually have messages that start with "Error using...."

A very common error

Error: File: debugging.m Line: 20 Column: 11
Incorrect use of '=' operator. To assign a value to a variable, use '='. To compare values for equality, use '=='.

In nearly all programming languages, you must distinguish between assignment of values to variables (`var = 2.5`) and comparison of variables (`if var == 2.5`). **Do not confuse == and =.** (Some languages, such as JavaScript, have a third operator, ===, to compare not only for the same value, but for identity of the objects that the variables refer to.)

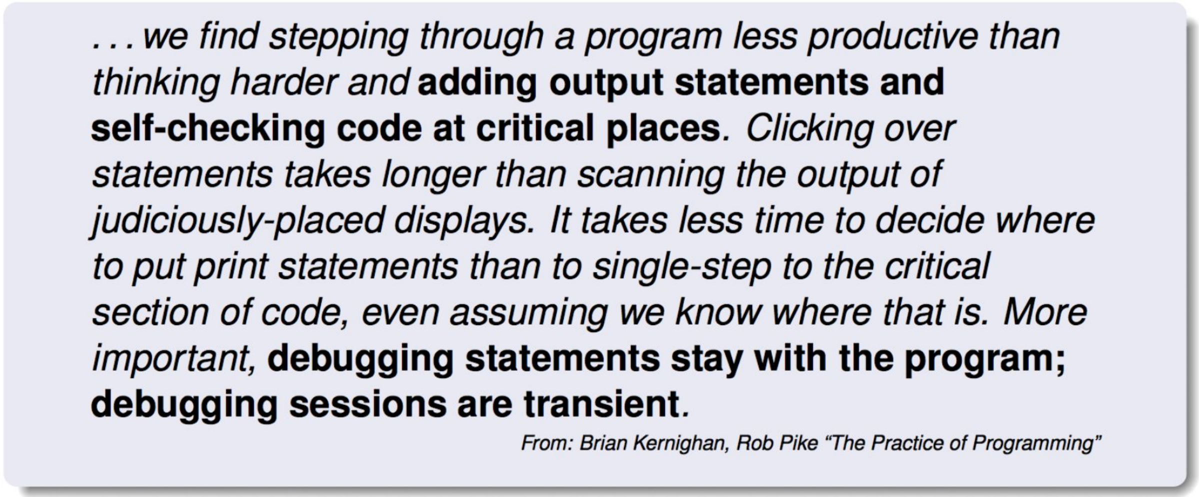
Summary of basic debugging techniques

- Read **error messages** very carefully
- Re-read code in the offending section
- Pay attention to the suggestions that MATLAB gives you, for example **red underlining and bars in the right margin**!
- For runtime errors: Inspect the variables in the **workspace**. Are all variables you need present? Do they have the right values?
- You can also just type the variable name in your **command interpreter** to find its value. Or use the class() function to find out more information.

For debugging complex code, or to find logical errors, you may need more powerful tools such as a **debugger.**

Intermediate debugging: Printing to output

In order to figure out what your program is doing, you can insert print / echo / fprintf / disp statements in your code to inspect the value of variables during the program run. With well instrumented programming languages such as MATLAB, this isn't usually the most effective way, but it is a common and valid method.



. . . we find stepping through a program less productive than thinking harder and **adding output statements and self-checking code at critical places**. *Clicking over statements takes longer than scanning the output of judiciously-placed displays. It takes less time to decide where to put print statements than to single-step to the critical section of code, even assuming we know where that is. More important,* **debugging statements stay with the program; debugging sessions are transient**.

From: Brian Kernighan, Rob Pike "The Practice of Programming"

Advanced debugging: Using a debugger!

Many programming languages come with debuggers, either as part of the language, the IDE or as a library you can load. MATLAB has a particularly easy-to-use debugger built in right into the IDE! The basic functions of a debugger are:

- Set breakpoints (= points in the execution of the program where it pauses): click on the right margin
- Step through the program line-by-line (expression-by-expression): use menu
- Inspect the variable values at each step: use workspace or command line

The debugger reference is here: https://www.mathworks.com/help/matlab/matlab_prog/debugging-process-and-features.html

Avoiding errors 1: use try-catch

When you can expect that errors might arise, you can/should! use the try-catch conditional expression to avoid your program crashing. The mechanism is called an exception.

Typical cases are:

- When opening a file (and you aren't sure it is actually present)
- When trying to retrieve data from a resource (that you don't know is available)

That is, when you rely on elements in your environment that may or may not be present.

Avoiding errors 2: Use testing

Testing is a key technique used by professional software development teams to find and reduce programming errors. You can and should use it, too! 2 types:

- Integration testing: Test the whole of your code by running it with known inputs, comparing the outputs with expectations (and fixing all crashes until the program runs without syntax and runtime errors).
- Unit testing: This means writing code that automatically "exercises" your program code. This usually requires to write your code in a modular way, with functions for each logical unit. For example, if part of your code contains a linear regression, put it in a function and write another function that tests the first function. **Rule of thumb: No code file should be > 100 lines of code.**