Fundamental Programming Principles: Variables and Functions

Beyond the Mouse

GEOS 436/636

Jeff Freymueller, Sep 12, 2017

"The Uncomfortable Truths Well", http://xkcd.com/568 (April 13, 2009)

Topics for Today

- A quick review of variables (Ch. 2)
- How MATLAB handles variables (Ch. 2)
- Advanced variable types (Ch. 11.4-11.5)
  - Cell arrays
  - Structures (structs)
- Functions (Ch. 6.1-6.3)
  - Built-in functions
  - Designing your own functions
  - Sensible use of functions

Variables Review: name vs. value

- Every variable has a name and a value – don't mix up the two.
  - The variable is a box in which you can store something. The name is written on the box, and the value is what you store inside.
  - Some languages have a few restricted words that cannot/should not be used for variable names, usually because these are the names of commands or control structures.
- MATLAB really treats all variable values, even strings, as arrays.

Variable review: assignment vs. reference

- Are you putting the value in, or taking it out?
- Assignment is when you store a value in a variable
  - `deg2rad = pi/180;`
- Reference is when you access the value.
  - "`pi`" above is a reference. It is replaced by the value of the variable called `pi`.
  - Some languages use a special symbol when you reference the value of a variable, but MATLAB does not

Audience Participation

For each statement, identify all variable assignments and references:

- `h = 6.62606896*10^-34;`
- `h_bar = h/(2*pi);`
- `(b == a_row(4))`
- `c = (a^2 + b^2)^0.5;`
- `j = j + 1;`

Variables in MATLAB

- MATLAB treats all variables as arrays (vectors or matrices). It's roots are in linear algebra.
  - Scalars are just 0-dimensional arrays (single values)
  - Values assigned using = : `a_row = [1 2 3]`
  - Values assigned using = : `a_col = [1; 2; 3]`
  - You can make an empty array: `foo = [];`
- Most of the time, you don't need to worry about the variable type, MATLAB handles it invisibly.
  - But you do have to remember that there is a difference between a row vector and a column vector.

Matrix, Row Vector, Column Vector

- A matrix $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
- A row vector $\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$
- A column vector $\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$
  - They are not the same (try it):

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = 1 + 4 + 9 = 14$$

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{bmatrix}$$

Array vs. Cell array

- Cell arrays are a special kind of array.
  - Every element of a regular array must be of the same type.
  - Not so for cell arrays. *Each element of a cell array is a container that can hold any one thing*.
  - Cell arrays are really useful for holding strings, and also are returned by some functions that read files.
  - They are useful for storing arrays of things that are not numbers
  - You can do numerical operations across regular arrays, but not cell arrays.

Bracketology

- MATLAB uses three different kinds of brackets, parentheses, braces, all meaning different things
- [ ] Square brackets
  - Vectors, arrays and matrices are contained inside
- ( ) Parentheses
  - Access a particular element of an array by putting the indices inside parentheses
- { } Braces or Curly brackets
  - Like parentheses, except for cell arrays

Bracketology

- [ ] Square brackets
  - Vectors, arrays and matrices are contained inside
  - `a_row = [1 2 3]; a_col = [1; 2; 3];`
  - `also_a_col = [1 2 3]';`
- ( ) Parentheses
  - Access a particular element of an array by putting the indices inside parentheses
  - What is the value of `a_row(2)`? of `a_col(3)`?
  - `a_row(3) = 5;`
- { } Braces or Curly brackets
  - Like parentheses, except for cell arrays
  - `my_cell{1} = 'Label';`
  - `my_cell{2} = [1 2 3];`

Math Operators

- +, -, *, .*, /, ./, ^, .^, \, .\
- Basic math operators (+, -, *, /, ^) operate on arrays.
  - * is actually matrix multiplication, / will invert a matrix
  - `a/b` is `a*inv(b)` while `a\b` is `inv(a)*b`
- Element-wise operators (.*, ./, .^, .\) operate on an element by element basis
  - `c = a.*b` means `c(i) = a(i)*b(i)`, for all `i`
  - Why is there no .+ nor .- ??
- Order of operators is normal math order. If you are not sure, use parentheses to force the order.
- `help ops` or `doc ops`

Array Expansion

- MATLAB will automatically expand the size of an array when you **assign** an element that does not exist
  - This is convenient when it is what you meant to do.
  - It causes trouble when you did not mean to do it.
  - It can be slow when arrays are big
- Example: `a = [1, 4]; a(2,1) = 5`
  - What do you think will happen?
- Example: `clear a; a = [1, 4]; c = a(2,1)`
  - What do you think will happen?

Special “Numbers”

- MATLAB handles complex numbers seamlessly
  - `5 + sqrt(-1)` evaluates to 5.0000 + 1.0000i
  - If you do not define a variable “i” or “j”, MATLAB will use either symbol for sqrt(-1).
- Not a Number (NaN) is a very handy “number”
  - Use NaN to represent missing values
    - DO NOT use “9999” for missing values!
  - Any arithmetic operation with NaN produces NaN
  - The function “isnan” finds all the NaNs in its argument
    - `idx = isnan(has_a_nan)`

Structures

- Suppose you have a set of variables that go together semantically. A structure (or struct) lets you package them together, making it easy to keep track of things.
- A ***struct*** has one or more ***fields***, which are named, and each can store a value (or vector, or array, or a struct, or …)
- You define a struct by naming its fields and assigning a value to each (or use [] for an empty value).
- Access a field like this: `weather.year`, `weather.temp(5)`
  - Or `getfield(weather, "year")` or `getfield(weather, name)` where `name` is a variable.

Struct Example 1

```
>> load st_elias
>> st_elias

st_elias =

        name: 'St. Elias region interpolated grid'
        type: 'velocity'
        unit: 'cm/yr'
        bbox: [2x2 double]
     timedep: []
     div_lon: 0.2500
     div_lat: 0.2500
    lonarray: [41x1 double]
    latarray: [27x1 double]
        east: [27x41 double]
       north: [27x41 double]
      height: [27x41 double]
```

The commands “load” and “save” let you store workspace or variables to disk. In this case, there is a file `st_elias.mat` that stores the saved variable `st_elias`.

This struct stores a gridded data set (in this case a model computation). It has some meta-data, a bounding box (`bbox`), information about the grid, and then data values.

Functions

- What is a function?
  - A set of mathematical operations that take some input values ("variables") and produce one or more output values.
    - Remember every value can be a scalar or a vector or matrix
  - A little black box of code that takes some inputs and produces one or more outputs
    - Some of the boxes in your flowchart might be implemented as functions
- Examples of built-in functions: **sqrt**, **isnan**, **find**, **eye**, **zeros**, **ones**, **size**, **inv**
- We introduced you to the use of some basic math in the last lab exercise, but functions are not limited to mathematical operations

Tasks That Might be Done by Functions

- Evaluate a mathematical function or perform an operation (like taking the spatial derivative of an array of numbers representing values on a grid).
- Compute several quantities derived from a set of data
- Read a data file that has a particular format, and perhaps do some basic processing on the values so that they are returned in a convenient form.
- Get input from the user on several questions, and return all of these answers.
- Any task that is repeated several times in your code, but with different inputs each time (like our `eat()` example in week 1).

Defining your own MATLAB functions

- Save your function in its own .m file, named for the function
- Start with a function declaration
  - `function output = combine(input1, input2)`
  - `function [diff12, diff23] = differences(in1, in2, in3)`
  - ***Be sure to give your function a descriptive name!***
- End with "**return**" (not required, but good habit)
- You can use any number of inputs and any number of outputs, as defined in your code.
  - The arguments to the function are ***passed by value***
    - `c = my_func(a, b);`
  - If you change the values of the input variables inside the function, those changes are lost when you exit
  - Some languages (like fortran) pass arguments by reference, so you can change any variable passed to a subroutine/function
- Any comment lines immediately after function definition are used by `help`

Not a descriptive name!

A simple function example

Let's make a "cuberoot" function.

- Create a file called cuberoot.m

```
function out = cuberoot(in);
out = in.^(1/3);
return
```

Why .^ ?

- Use the function
  - `y = cuberoot(x);`
- What's going on when you call the function
  - MATLAB takes the value of x, and assigns it to the variable in on the inside of the function
  - When the function exits, the value of out is assigned to y

Scope of variables

- The function exists in its own separate little space. It interacts with its calling routine ***only*** through the arguments it is passed and the values it returns.
  - It can only modify the values it returns, and not the arguments.
- Variables used inside the function are created when the function is called, and thrown out when it is done.
- You can re-use variable names inside a function that are also used somewhere outside.

An Example

```
function [diff12, diff23] = differences(in1, in2, in3)
    diff12 = in2 – in1;
    diff23 = in3 – in2;
return
```

File: differences.m

- Here is a sample line of code calling the function:
  - `[d1, d2] = differences(3, 4, 6)`
- So what happens inside the function?
  - Values are assigned to the variables `in1`, `in2`, `in3`. Each variable gets ***a copy of the input arguments***.
  - The function returns values for two output arguments, which it internally calls `diff12` and `diff23`.
- What happens on the outside?
  - The input arguments (three numbers here) cannot be changed
  - The variable `d1` is assigned the first output value and `d2` the second output value.

More of your own functions

- As long as MATLAB knows where to find it, your function becomes just as much a part of the language as the built-in functions
  - Build up a set of your own useful functions!
    - MATLAB will always find functions in the current directory
    - Use `addpath` to specify other directories where it should look for your functions
  - Assemble small pieces into bigger tools!
- Be sure to use sensible names!