Lec11 IAM550 J. Raeder 10/01/2019 Structured programming, functions

- Rehash loops, assignments in vectors.
- Rehash sprintf(), print formats, construct file names.

- For larger projects writing a single code (unstructured coding) in a single file becomes impractical and error prone → think of sin() function. Multiple arguments → atan2(y,x).
- Functions allow to break the code into subtasks that can be separately programmed and tested.
- Functions also allow to avoid repeated code. Example: built-in functions like sin(), cos(), exp(), log(), ⋯
- We already used *inline functions* (in Fortran *statement functions*, in C *#define macros*) but they are basically limited to one-liners.
- Unfortunately, MATLAB handles functions in an awkward way (still better than no functions).
- To start with, define functions within a main script.

```
x = 1:10;
n = length(x);
avg = mymean(x,n);
med = mymedian(x,n);

function a = mymean(v,n)
% MYMEAN Example of a local function.

    a = sum(v)/n;
end

function m = mymedian(v,n)
% MYMEDIAN Another example of a local function.

    w = sort(v);
    if rem(n,2) == 1  %  remainder
        m = w((n + 1)/2);
    else
        m = (w(n/2) + w(n/2 + 1))/2;
    end
end
```

- A function is declared by the `function` keyword.
- The function ends either with `end`, another `function` definition, or the end of the file. Using `end` is preferred.
- The function may or may not have a return value. (with no return value: C: void funcname(⋯); Fortran: subroutine funcname(⋯)).
- A function may or may not (not makes not so much sense in MATLAB) have arguments.
- In the function itself the arguments are called `dummy arguments`.
- When the function is called they are called `actual arguments.`

- Matlab passes arguments by value: the `actual arguments` are copied into the `dummy arguments.` Thus, when the dummy arguments are modified, the actual arguments are not modified.
- The other method is called ‘passing arguments by reference’ or ‘passing by pointer’. In that case, nothing is copied, but only a memory reference is passed and changing a value in the function will also change the value in the calling program (which is also a way ro return results)
- The `actual arguments` and the `dummy arguments` can have different names, but they must match in type.
- There can be fewer `actual arguments` than `dummy arguments`, but unmatched arguments better not be used. There cannot be more actual than dummy arguments.
- The number of arguments in the calling statement is known by `nargin` and `nargout.`
- Multiple values can be returned:

```
%  in calling program:
[c_abs c_emm] = cross(1830);

function [cross_abs, cross_emm] = cross(lambda)
%This function is used to calculate the absorption cross section
%and emmision cross section for the given wavelength lambda
%Note that the unit of the lambda here should be nm
nargin
nargout   %  number of in/out arguments
load absorption_cross.txt;
load emission_cross.txt;
%  interpolate from tables, see
%  https://www.mathworks.com/help/matlab/ref/interp1.html
cross_abs=interp1(absorption_cross(:,1),absorption_cross(:,2),lam
bda,'spline')*1e-24;
cross_emm=interp1(emission_cross(:,1),emission_cross(:,2),lambda,
'spline')*1e-24;
end
```

- The return arguments are a list. Like with the actual/dummy arguments, the names do not need to match, but the types.
- Return arguments are also passed by value, that is, copied from the function back to the calling program.
- The variables in the function are `local`, that is, a variable with the same name in the function as in the main program will not be changed in one unit if it is changed in another. Fortran, C: local; Perl: always global, unless specified otherwise.

- Next time: global vars, persistent vars, function m-files, recursion, hierarchy of functions, programming with functions