Common Errors

- When adding new runtime parameters
  - Add to Config, Unit_data module, AND Unit_init
  - use 'use' Unit_data
  - Re-setup and run, looking for warning messages in stdout
- Use “use Unit_interface, ONLY:” for all top level subroutines
- When iterating over blocks do not confuse the list index with the blockID
- When writing your own refinement criteria, be careful not to override something that should forbid refinement.

  In your implementation based on gr_markRefineDerefine.F90:
  - refine(ib) = .true.  ! OK
  - derefine(ib) = .false.  ! OK
  - derefine(ib) = .true.  ! Probably OK...
  - refine(ib) = .false.  ! Maybe trouble, if refine(ib) was set!
Good ideas

- Pay attention to warnings
  - Tempting to ignore sometimes, but more often than not trouble
  - They may not always stand out in the output, look for them
- Use all available output to analyze a problem
  - setup_* files often have useful information
  - flash.dat can often be very telling
  - The logfile and stdout, not all messages are written to the logfile!
  - Adding per-processor logfiles can help a lot
  - Analyze binary files with both a visualization tool, and binary utilities (ex. h5dump)
More good ideas

- Check the AMR grid for sanity
  - Is the refinement pattern valid?
  - Is it what you expect?
  - Check the corners, too!
- When testing a setup use `-noclober`
  - Unless you are making very basic changes to the problem, this can save a lot of build time
A physics unit is responsible for making sure that guard cells are filled
- A physics unit, however, does not have to update guard cells before returning

Each physics unit is responsible for leaving solution data in a thermodynamically consistent state before returning

Alternative implementations for units should go under the stubs they implement
- Implement a source/Eos/EosMain/MyEos/… rather than source/MyEos/…
Sfocu and you

- Sfocu -- Serial Flash Output Comparison Utility
- Performs a block-by-block comparison of data
- Will also compare particle-to-particle
- Will attempt to find the best match if there is a mismatch in grid structure
- By default, the test fails if there is any discrepancy between the two files, but an error tolerance can be added.
- Great for catching small errors that are too small to be detected via visualization
- Can also check face-centered data by using the -s (self-discovery) flag.
- Shows the size of the error seen as well as a normalized error
As Flash becomes more complex, the ability to easily add new test cases has become important.

Primarily used for regression testing:
- Compare the results of a run against the results of a run that was known to be correct
- All tests except unit tests are fundamentally regression tests

Can be invoked as needed:
- `flashTest.py [options] -f <jobsFile>`

Relies on a *test.info* file for information on configuring flash:
- All tests require their own setup line, a parfile, and information on which files to compare

Requires a working version of sfocu for any of the regression tests.

Freely available from the Flash website.
FlashTest - Test Types

- **UnitTest**
  - Set up and invoke a Flash unit test. Looks for files that contain the line “all results conformed with expected values.”
  - If there is an error, putting in debugging information can be very helpful.
FlashTest - Test Types

- **Comparison Tests**
  - Compare to yesterday’s result and to a known working benchmark

- **Restart Tests**
  - Run a Flash problem creating at least two checkpoints. Restart from the intermediate checkpoint and compare the two end files
FlashTest -- Test Types

- Composite Test
  - New type of test added that combines a comparison and restart test
  - Instead of doing a regression to yesterday, compares to the last time the comparison benchmark changed
Pairs with Flash Test

Provides a web-based interface to quickly summarize results and to ease checking and updating benchmarks