Some Programming Principles

What is good code? It is worth thinking (and talking) about before we even begin to analyze, so take this as a spoiler alert. I could tell you what good code is, but you will grow your own aesthetic, and it will change over time. Many desirable properties of good code are in conflict with other desirable properties. An excellent piece of code in one context may be a weak link or a silly extravagance in another. That said, there are some properties generally considered worth attending to. In no particular order (one should *not* prioritize such lists), here are 7:

fast, small, general, portable, accurate, clear, robust

ATBE (all things being equal), my code should run fast. If it is part of an antilock brake system, it must run fast, but a fraction of a second in a 5 minute simulation goes unnoticed. Silly code notwithstanding, speed is fundamentally a function of algorithm, language and hardware.

ATBE (but of course they are not), my code should be small (use little memory, here I mean data storage, not how many lines of code). Memory near the registers is expensive and fast, and while memory gets cheaper as we move away, it gets much slower. Furthermore, the code I am writing is almost certainly only one piece of a process which will compete for memory. The amount of available memory is often an active contraint in algorithm choice.

ATBEBOCTAN, my code should be general (solve many instances and variations of the problem). One can always make a general purpose routine go faster by specializing the type of problem it can solve, but it may be more useful if it can solve a wider class of problem.

ATBEBOCTAN, my code should be portable (run on many different machines with little rewriting). Taking advantages of properties of special computers can make a code run much faster, but reduces its portablilty.

ATBEBOCTAN, my code should be accurate (give the correct answers). We will have *a lot* to say about this, but for now let's just say that while we don't even hope to get the exact answer, we sometimes hope to get a best possible approximation.

ATBEBOCTAN, my code should be clear (well documented, and easy to understand, adapt, and incorporate into other code). We often make small sacrifices in speed and size for the sake of clarity. We should *always* make whatever sacrifice is necessary to communicate to the user how to interpret all input and output variables. Again, we note that our code is often one piece of some bigger project.

ATBEBOCTAN, my code should be robust (not fail unless it must, and tell me about the failure if it does). Here failure can be subtle or dramatic, from not achieving the accuracy that the data warrants, to not giving a solution at all. It is very bad for a program to crash and possibly ruin many expensive computations, but it is usually worse for a code to fail and not tell us about it.