Codes for Assignment 6

For this assignment you are given two Python modules to work from. These illustrate features of Python relevant for the assignment. Here is some explanation of the features of these modules, which are common ways to do things in Python.

FakeData.py
This makes several sets of fake data, each going to its own file. The file names are made by *catenating* (the correct word in English is "concatenating", which means putting one after the other). For example, line 70 makes a name by putting together (catenating) a fixed string `"info_"` with the name of the run and then the `".py"` ending to set the filetype of this file. If the name of the run is `run1`, the catenation gives

$$\texttt{info\_} + \texttt{run1} + \texttt{.py} \implies \texttt{info\_run1.py} \quad .$$

The individual runs are defined in lines 19 to 44. Each run produces an info file and data file that contain the name of the run.

Lines 26, 35, and 44 put the parameters for a run into a Python *dictionary*. Other names for the dictionary data structure are *associative array* (in C++) and *hash table* (the generic name). Lines 50 to 55 show that a dictionary is like an array except that the index is not a number (as for an array) but the name of the object. Line 26 shows that you define a dictionary using "curleys" (curley braces, $\{\cdots\}$). Inside goes the name (as a string), a colon, and the value associated to that name. For example, the first part `"n":n` associates the name `n` with the value (which is 50 in this case). The Python documentation has more information about dictionaries. They are a convenient way to store information about some object, run objects in this case.

Line 48 shows how to use the array of run objects. Lines 50 to 55 "unpack" the run information. A different way to code this would have been to define a function that creates a fake data set and to call this function ones for each run.

modelFit.py
This module gets the name of its data files on the *command line*. You type: `python modelFit.py run1` to run `modelFit.py` on the files associated with `run1`. The `run1` in the command is a *command line argument*. The Python package `sys` (for "system") has a method that gives the module access to command line arguments, through the array[1] `argv`. The first element of `argv`, which is `sys.argv[0]`, is the name of the module (`modelFit.py` in this case). The

[1] This way of handling command line arguments comes from C, which has the variable `argc`, for the "count", or number of arguments, and `argv` for the "values" of the arguments.

next element, which is `sys.argv[1]`, is the first command line argument. This module has only one command line argument, but you could have more. Line 19 accesses this argument. Line 16 imports the `sys` package needed to do this.

Lines 17 and 26 imports a file whose name is not known when you write the module. The package `importlib` includes functions that give you more control over the import process.[2] If you had known the name `infoImportFile` in advance (`info_run1` for example) you could have done this in the simpler way: `import info_run1 as info`. Line 26 does the import, which means executing the commands in the file `infoImportFile` and putting the resulting names in the namespace `info`. Line 27 illustrates that this has happened. The module whose name is `infoImportFile` (with `.py` at the end) has a command `n = ...`. This creates (or re-purposes) the name `n` and binds it to the given value. The name goes into the `info` namespace.

[2]Warning: I found the official documentation for `importlib` too confusing to use by itself. Line 26 is copied from an example from another source.