The following scripts demonstrate the implementations of various Python modules.

The modules explored include;

bisect : Array bisection algorithm. This module provides support for maintaining a list in sorted order without having to sort the list after each insertion. For long lists of items with expensive comparison operations, this can be an improvement over the more common approach. The module is called bisect because it uses a basic bisection algorithm to do its work. The bisect() function can be useful for numeric table lookups. This example uses bisect() to look up a letter grade for an exam score (say) based on a set of ordered numeric breakpoints: 90 and up is an ‘A’, 80 to 89 is a ‘B’, and so on:

shlex: Simple lexical analysis. The shlex class makes it easy to write lexical analyzers for simple syntaxes resembling that of the Unix shell. This will often be useful for writing minilanguages, (for example, in run control files for Python applications) or for parsing quoted strings.

readline: GNU readline interface. The readline module defines a number of functions to facilitate completion and reading/writing of history files from the Python interpreter. This module can be used directly, or via the rlcompleter module, which supports completion of Python identifiers at the interactive prompt. Settings made using this module affect the behaviour of both the interpreter’s interactive prompt and the prompts offered by the built-in input() function.

operator: Standard operators as functions. The operator module exports a set of efficient functions corresponding to the intrinsic operators of Python. For example, operator.add(x, y) is equivalent to the expression x+y. Many function names are those used for special methods, without the double underscores. For backward compatibility, many of these have a variant with the double underscores kept. The variants without the double underscores are preferred for clarity. The functions fall into categories that perform object comparisons, logical operations, mathematical operations and sequence operations.

stat: Interpreting stat() results. The stat module defines constants and functions for interpreting the results of os.stat(), os.fstat() and os.lstat() (if they exist). Normally, you would use the os.path.is\*() functions for testing the type of a file; the functions here are useful when you are doing multiple tests of the same file and wish to avoid the overhead of the stat() system call for each test. These are also useful when checking for information about a file that isn’t handled by os.path, like the tests for block and character devices.

Compiled and presented by Vakindu Philliam.