Python Tutorial

1. Whetting Your Appetite

If you ever wrote a large shell script, you probably know this feeling: you'd love to add yet another feature, but it's already so slow, and so big, and so complicated; or the feature involves a system call or other function that is only accessible from C ...Usually the problem at hand isn't serious enough to warrant rewriting the script in C; perhaps the problem requires variable-length strings or other data types (like sorted lists of file names) that are easy in the shell but lots of work to implement in C, or perhaps you're not sufficiently familiar with C.

1.1. Where From Here

Now that you are all excited about Python, you'll want to examine it in some more detail. Since the best way to learn a language is using it, you are invited here to do so.

2. Using the Python Interpreter

2.1. Invoking the Interpreter

The interpreter's line-editing features usually aren't very sophisticated. On Unix, whoever installed the interpreter may have enabled support for the GNU readline library, which adds more elaborate interactive editing and history features. Perhaps the quickest check to see whether command line editing is supported is typing Control-P to the first Python prompt you get. If it beeps, you have command line editing; see Appendix A for an introduction to the keys. If nothing appears to happen, or if P is echoed, command line editing isn't available; you'll only be able to use backspace to remove characters from the current line.

2.1.1. Argument Passing

When known to the interpreter, the script name and additional arguments thereafter are passed to the script in the variable sys.argv, which is a list of strings. Its length is at least one; when no script and no arguments are given, sys.argv[0] is an empty string. When the script name is given as '-' (meaning standard input), sys.argv[0] is set to '-'. When -c command is used, sys.argv[0] is set to '-c'. Options found after -c command are not consumed by the Python interpreter's option processing but left in sys.argv for the command to handle.

2.1.2. Interactive Mode

When commands are read from a tty, the interpreter is said to be in interactive mode. In this mode it prompts for the next command with the primary prompt, usually three greater-than signs (">>> "); for continuation lines it prompts with the secondary prompt, by default three dots ("... ").

Continuation lines are needed when entering a multi-line construct.

2.2. The Interpreter and Its Environment

When an error occurs, the interpreter prints an error message and a stack trace. In interactive mode, it then returns to the primary prompt; when input came from a file, it exits with a nonzero exit status after printing the stack trace. (Exceptions handled by an except clause in a try statement are not errors in this context.) Some errors are unconditionally fatal and cause an exit with a nonzero exit; this applies to internal inconsistencies and some cases of running out of memory. All error messages are written to the standard error stream; normal output from the executed commands is written to standard output.

Typing the interrupt character (usually Control-C or DEL) to the primary or secondary prompt cancels the input and returns to the primary prompt.2.1Typing an interrupt while a command is executing raises the KeyboardInterrupt exception, which may be handled by a try statement.