

9. Top-level components

The Python interpreter can get its input from a number of sources: from a script passed to it as standard input or as program argument, typed in interactively, from a module source file, etc. This chapter gives the syntax used in these cases.

9.1. Complete Python programs

While a language specification need not prescribe how the language interpreter is invoked, it is useful to have a notion of a complete Python program. A complete Python program is executed in a minimally initialized environment: all built-in and standard modules are available, but none have been initialized, except for `sys` (various system services), `builtins` (built-in functions, exceptions and `None`) and `__main__`. The latter is used to provide the local and global namespace for execution of the complete program.

The syntax for a complete Python program is that for file input, described in the next section.

The interpreter may also be invoked in interactive mode; in this case, it does not read and execute a complete program but reads and executes one statement (possibly compound) at a time. The initial environment is identical to that of a complete program; each statement is executed in the namespace of `__main__`.

Under Unix, a complete program can be passed to the interpreter in three forms: with the `-c` *string* command line option, as a file passed as the first command line argument, or as standard input. If the file or standard input is a tty device, the interpreter enters interactive mode; otherwise, it executes the file as a complete program.

9.2. File input

All input read from non-interactive files has the same form:

`file_input ::= (NEWLINE | statement)*`

This syntax is used in the following situations:

- when parsing a complete Python program (from a file or from a string);
- when parsing a module;
- when parsing a string passed to the `exec()` function;

9.3. Interactive input

Input in interactive mode is parsed using the following grammar:

```
interactive_input ::= [stmt_list] NEWLINE | compound_stmt NEWLINE
```

Note that a (top-level) compound statement must be followed by a blank line in interactive mode; this is needed to help the parser detect the end of the input.

9.4. Expression input

`eval()` is used for expression input. It ignores leading whitespace. The string argument to `eval()` must have the following form:

```
eval_input ::= expression_list NEWLINE*
```