11.3. `fileinput` — Iterate over lines from multiple input streams

Source code: Lib/fileinput.py

This module implements a helper class and functions to quickly write a loop over standard input or a list of files. If you just want to read or write one file see `open()`.

The typical use is:

```
import fileinput
for line in fileinput.input():
    process(line)
```

This iterates over the lines of all files listed in `sys.argv[1:]`, defaulting to `sys.stdin` if the list is empty. If a filename is `'-'`, it is also replaced by `sys.stdin`. To specify an alternative list of filenames, pass it as the first argument to `input()`. A single file name is also allowed.

All files are opened in text mode by default, but you can override this by specifying the *mode* parameter in the call to `input()` or `FileInput`. If an I/O error occurs during opening or reading a file, `OSError` is raised.

Changed in version 3.3: `IOError` used to be raised; it is now an alias of `OSError`.

If `sys.stdin` is used more than once, the second and further use will return no lines, except perhaps for interactive use, or if it has been explicitly reset (e.g. using `sys.stdin.seek(0)`).

Empty files are opened and immediately closed; the only time their presence in the list of filenames is noticeable at all is when the last file opened is empty.

Lines are returned with any newlines intact, which means that the last line in a file may not have one.

You can control how files are opened by providing an opening hook via the *openhook* parameter to `fileinput.input()` or `FileInput()`. The hook must be a function that takes two arguments, *filename* and *mode*, and returns an accordingly opened file-like object. Two useful hooks are already provided by this module.

The following function is the primary interface of this module:

`fileinput.`**`input`**(*files=None*, *inplace=False*, *backup=''*, *bufsize=0*, *mode='r'*, *openhook=None*)

Create an instance of the `FileInput` class. The instance will be used as global state for the functions of this module, and is also returned to use during iteration. The parameters to this function will be passed along to the constructor of the `FileInput` class.

The `FileInput` instance can be used as a context manager in the `with` statement. In this example, *input* is closed after the `with` statement is exited, even if an exception occurs:

```
with fileinput.input(files=('spam.txt', 'eggs.txt')) as f:
    for line in f:
        process(line)
```

Changed in version 3.2: Can be used as a context manager.

Deprecated since version 3.6, will be removed in version 3.8: The *bufsize* parameter.

The following functions use the global state created by `fileinput.input()`; if there is no active state, `RuntimeError` is raised.

`fileinput.`**`filename`**()
Return the name of the file currently being read. Before the first line has been read, returns `None`.

`fileinput.`**`fileno`**()
Return the integer “file descriptor” for the current file. When no file is opened (before the first line and between files), returns `-1`.

`fileinput.`**`lineno`**()
Return the cumulative line number of the line that has just been read. Before the first line has been read, returns `0`. After the last line of the last file has been read, returns the line number of that line.

`fileinput.`**`filelineno`**()
Return the line number in the current file. Before the first line has been read, returns `0`. After the last line of the last file has been read, returns the line number of that line within the file.

`fileinput.`**`isfirstline`**()
Returns true if the line just read is the first line of its file, otherwise returns false.

`fileinput.`**`isstdin`**()
Returns true if the last line was read from `sys.stdin`, otherwise returns false.

`fileinput.`**`nextfile`**()

Close the current file so that the next iteration will read the first line from the next file (if any); lines not read from the file will not count towards the cumulative line count. The filename is not changed until after the first line of the next file has been read. Before the first line has been read, this function has no effect; it cannot be used to skip the first file. After the last line of the last file has been read, this function has no effect.

`fileinput.`**`close`**()

Close the sequence.

The class which implements the sequence behavior provided by the module is available for subclassing as well:

class `fileinput.`**`FileInput`**(*files=None*, *inplace=False*, *backup=''*, *bufsize=0*, *mode='r'*, *openhook=None*)

Class `FileInput` is the implementation; its methods `filename()`, `fileno()`, `lineno()`, `filelineno()`, `isfirstline()`, `isstdin()`, `nextfile()` and `close()` correspond to the functions of the same name in the module. In addition it has a `readline()` method which returns the next input line, and a `__getitem__()` method which implements the sequence behavior. The sequence must be accessed in strictly sequential order; random access and `readline()` cannot be mixed.

With *mode* you can specify which file mode will be passed to `open()`. It must be one of `'r'`, `'rU'`, `'U'` and `'rb'`.

The *openhook*, when given, must be a function that takes two arguments, *filename* and *mode*, and returns an accordingly opened file-like object. You cannot use *inplace* and *openhook* together.

A `FileInput` instance can be used as a context manager in the `with` statement. In this example, *input* is closed after the `with` statement is exited, even if an exception occurs:

```
with FileInput(files=('spam.txt', 'eggs.txt')) as input:
    process(input)
```

Changed in version 3.2: Can be used as a context manager.

Deprecated since version 3.4: The `'rU'` and `'U'` modes.

Deprecated since version 3.6, will be removed in version 3.8: The *bufsize* parameter.

Optional in-place filtering: if the keyword argument `inplace=True` is passed to `fileinput.input()` or to the `FileInput` constructor, the file is moved to a backup file and standard output is directed to the input file (if a file of the same name as the backup file already exists, it will be replaced silently). This makes it possible to write a filter that rewrites its input file in place. If the *backup* parameter is given (typically as `backup='.<some extension>'`), it specifies the extension for the backup file, and the backup file remains around; by default, the extension is `'.bak'` and it is deleted when the output file is closed. In-place filtering is disabled when standard input is read.

The two following opening hooks are provided by this module:

`fileinput.`**`hook_compressed`**(*filename*, *mode*)

Transparently opens files compressed with gzip and bzip2 (recognized by the extensions `'.gz'` and `'.bz2'`) using the `gzip` and `bz2` modules. If the filename extension is not `'.gz'` or `'.bz2'`, the file is opened normally (ie, using `open()` without any decompression).

Usage example: `fi = fileinput.FileInput(openhook=fileinput.hook_compressed)`

`fileinput.`**`hook_encoded`**(*encoding*, *errors=None*)

Returns a hook which opens each file with `open()`, using the given *encoding* and *errors* to read the file.

Usage example: `fi = fileinput.FileInput(openhook=fileinput.hook_encoded("utf-8", "surrogateescape"))`

Changed in version 3.6: Added the optional *errors* parameter.