11.8. `fnmatch` — Unix filename pattern matching

Source code: Lib/fnmatch.py

This module provides support for Unix shell-style wildcards, which are *not* the same as regular expressions (which are documented in the `re` module). The special characters used in shell-style wildcards are:

Pattern	Meaning
`*`	matches everything
`?`	matches any single character
`[seq]`	matches any character in *seq*
`[!seq]`	matches any character not in *seq*

For a literal match, wrap the meta-characters in brackets. For example, `'[?]'` matches the character `'?'`.

Note that the filename separator (`'/'` on Unix) is *not* special to this module. See module `glob` for pathname expansion (`glob` uses `fnmatch()` to match pathname segments). Similarly, filenames starting with a period are not special for this module, and are matched by the `*` and `?` patterns.

`fnmatch.`**`fnmatch`**(*filename*, *pattern*)

Test whether the *filename* string matches the *pattern* string, returning `True` or `False`. Both parameters are case-normalized using `os.path.normcase()`. `fnmatchcase()` can be used to perform a case-sensitive comparison, regardless of whether that's standard for the operating system.

This example will print all file names in the current directory with the extension `.txt`:

```
import fnmatch
import os

for file in os.listdir('.'):
    if fnmatch.fnmatch(file, '*.txt'):
        print(file)
```

`fnmatch.`**`fnmatchcase`**(*filename*, *pattern*)

Test whether *filename* matches *pattern*, returning `True` or `False`; the comparison is case-sensitive and does not apply `os.path.normcase()`.

`fnmatch.`**`filter`**(*names*, *pattern*)

Return the subset of the list of *names* that match *pattern*. It is the same as `[n for n in names if fnmatch(n, pattern)]`, but implemented more efficiently.

`fnmatch.`**`translate`**(*pattern*)

Return the shell-style *pattern* converted to a regular expression for using with `re.match()`.

Example:

```
>>> import fnmatch, re
>>>
>>> regex = fnmatch.translate('*.txt')
>>> regex
'(?s:.*\\.txt)\\Z'
>>> reobj = re.compile(regex)
>>> reobj.match('foobar.txt')
<_sre.SRE_Match object; span=(0, 10), match='foobar.txt'>
```

See also:

Module `glob`

Unix shell-style path expansion.