`glob` — Unix style pathname pattern expansion

Source code: Lib/glob.py

The `glob` module finds all the pathnames matching a specified pattern according to the rules used by the Unix shell, although results are returned in arbitrary order. No tilde expansion is done, but `*`, `?`, and character ranges expressed with `[]` will be correctly matched. This is done by using the `os.scandir()` and `fnmatch.fnmatch()` functions in concert, and not by actually invoking a subshell.

Note that files beginning with a dot (`.`) can only be matched by patterns that also start with a dot, unlike `fnmatch.fnmatch()` or `pathlib.Path.glob()`. (For tilde and shell variable expansion, use `os.path.expanduser()` and `os.path.expandvars()`.)

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

The `glob` module defines the following functions:

`glob.glob(pathname, *, root_dir=None, dir_fd=None, recursive=False, include_hidden=False)`

Return a possibly empty list of path names that match *pathname*, which must be a string containing a path specification. *pathname* can be either absolute (like `/usr/src/Python-1.5/Makefile`) or relative (like `../../Tools/*/*.gif`), and can contain shell-style wildcards. Broken symlinks are included in the results (as in the shell). Whether or not the results are sorted depends on the file system. If a file that satisfies conditions is removed or added during the call of this function, whether a path name for that file will be included is unspecified.

If *root_dir* is not `None`, it should be a path-like object specifying the root directory for searching. It has the same effect on `glob()` as changing the current directory before calling it. If *pathname* is relative, the result will contain paths relative to *root_dir*.

This function can support paths relative to directory descriptors with the *dir_fd* parameter.

If *recursive* is true, the pattern "`**`" will match any files and zero or more directories, subdirectories and symbolic links to directories. If the pattern is followed by an `os.sep` or `os.altsep` then files will not match.

If *include_hidden* is true, "`**`" pattern will match hidden directories.

Raises an auditing event `glob.glob` with arguments `pathname`, `recursive`.

Raises an auditing event `glob.glob/2` with arguments `pathname`, `recursive`, `root_dir`, `dir_fd`.

Note: Using the "`**`" pattern in large directory trees may consume an inordinate amount of time.

Note: This function may return duplicate path names if *pathname* contains multiple "**" patterns and *recursive* is true.

Changed in version 3.5: Support for recursive globs using "**".

Changed in version 3.10: Added the *root_dir* and *dir_fd* parameters.

Changed in version 3.11: Added the *include_hidden* parameter.

glob.**iglob**(*pathname*, ***, *root_dir=None*, *dir_fd=None*, *recursive=False*, *include_hidden=False*)

Return an iterator which yields the same values as `glob()` without actually storing them all simultaneously.

Raises an auditing event `glob.glob` with arguments `pathname`, `recursive`.

Raises an auditing event `glob.glob/2` with arguments `pathname`, `recursive`, `root_dir`, `dir_fd`.

Note: This function may return duplicate path names if *pathname* contains multiple "**" patterns and *recursive* is true.

Changed in version 3.5: Support for recursive globs using "**".

Changed in version 3.10: Added the *root_dir* and *dir_fd* parameters.

Changed in version 3.11: Added the *include_hidden* parameter.

glob.**escape**(*pathname*)

Escape all special characters (`'?'`, `'*'` and `'['`). This is useful if you want to match an arbitrary literal string that may have special characters in it. Special characters in drive/UNC sharepoints are not escaped, e.g. on Windows `escape('//?/c:/Quo vadis?.txt')` returns `'//?/c:/Quo vadis[?].txt'`.

Added in version 3.4.

glob.**translate**(*pathname*, ***, *recursive=False*, *include_hidden=False*, *seps=None*)

Convert the given path specification to a regular expression for use with `re.match()`. The path specification can contain shell-style wildcards.

For example:

```
>>> import glob, re
>>>
>>> regex = glob.translate('**/*.txt', recursive=True, include_hidden=True)
>>> regex
'(?s:(?:.+/)?[^/]*\\.txt)\\Z'
>>> reobj = re.compile(regex)
>>> reobj.match('foo/bar/baz.txt')
<re.Match object; span=(0, 15), match='foo/bar/baz.txt'>
```

Path separators and segments are meaningful to this function, unlike `fnmatch.translate()`. By default wildcards do not match path separators, and `*` pattern segments match precisely one path segment.

If *recursive* is true, the pattern segment "`**`" will match any number of path segments.

If *include_hidden* is true, wildcards can match path segments that start with a dot (`.`).

A sequence of path separators may be supplied to the *seps* argument. If not given, `os.sep` and `altsep` (if available) are used.

See also: `pathlib.PurePath.full_match()` and `pathlib.Path.glob()` methods, which call this function to implement pattern matching and globbing.

Added in version 3.13.

Examples

Consider a directory containing the following files: `1.gif`, `2.txt`, `card.gif` and a subdirectory `sub` which contains only the file `3.txt`. `glob()` will produce the following results. Notice how any leading components of the path are preserved.

```
>>> import glob
>>> glob.glob('./[0-9].*')
['./1.gif', './2.txt']
>>> glob.glob('*.gif')
['1.gif', 'card.gif']
>>> glob.glob('?.gif')
['1.gif']
>>> glob.glob('**/*.txt', recursive=True)
['2.txt', 'sub/3.txt']
>>> glob.glob('./**/', recursive=True)
['./', './sub/']
```

If the directory contains files starting with `.` they won't be matched by default. For example, consider a directory containing `card.gif` and `.card.gif`:

```
>>> import glob
>>> glob.glob('*.gif')
['card.gif']
>>> glob.glob('.c*')
['.card.gif']
```

See also: The `fnmatch` module offers shell-style filename (not path) expansion.

See also: The `pathlib` module offers high-level path objects.