glob — Filename Pattern Matching

Purpose: Use Unix shell rules to find filenames matching a pattern.

Even though the `glob` API is small, the module packs a lot of power. It is useful in any situation where a program needs to look for a list of files on the file system with names matching a pattern. To create a list of filenames that all have a certain extension, prefix, or any common string in the middle, use `glob` instead of writing custom code to scan the directory contents.

The pattern rules for `glob` are not the same as the regular expressions used by the re module. Instead, they follow standard Unix path expansion rules. There are only a few special characters used to implement two different wild-cards and character ranges. The pattern rules are applied to segments of the filename (stopping at the path separator, `/`). Paths in the pattern can be relative or absolute. Shell variable names and tilde (~) are not expanded.

Example Data

The examples in this section assume the following test files are present in the current working directory.

```
$ python3 glob_maketestdata.py

dir
dir/file.txt
dir/file1.txt
dir/file2.txt
dir/filea.txt
dir/fileb.txt
dir/file?.txt
dir/file*.txt
dir/file[.txt
dir/subdir
dir/subdir/subfile.txt
```

If these files do not exist, use `glob_maketestdata.py` in the sample code to create them before running the following examples.

Wildcards

An asterisk (*) matches zero or more characters in a segment of a name. For example, `dir/*`.

```
# glob_asterisk.py

import glob
for name in sorted(glob.glob('dir/*')):
    print(name)
```

The pattern matches every path name (file or directory) in the directory dir, without recursing further into subdirectories. The data returned by `glob()` is not sorted, so the examples here sort it to make studying the results easier.

```
$ python3 glob_asterisk.py

dir/file*.txt
dir/file.txt
dir/file1.txt
dir/file2.txt
dir/file?.txt
dir/file[.txt
dir/filea.txt
dir/fileb.txt
dir/subdir
```

To list files in a subdirectory, the subdirectory must be included in the pattern.

```
# glob_subdir.py
```

```
import glob

print('Named explicitly:')
for name in sorted(glob.glob('dir/subdir/*')):
    print('  {}'.format(name))

print('Named with wildcard:')
for name in sorted(glob.glob('dir/*/*')):
    print('  {}'.format(name))
```

The first case shown earlier lists the subdirectory name explicitly, while the second case depends on a wildcard to find the directory.

```
$ python3 glob_subdir.py

Named explicitly:
  dir/subdir/subfile.txt
Named with wildcard:
  dir/subdir/subfile.txt
```

The results, in this case, are the same. If there was another subdirectory, the wildcard would match both subdirectories and include the filenames from both.

Single Character Wildcard

A question mark (?) is another wildcard character. It matches any single character in that position in the name.

```
# glob_question.py
```

```
import glob

for name in sorted(glob.glob('dir/file?.txt')):
    print(name)
```

The previous example matches all of the filenames that begin with `file`, have one more character of any type, then end with `.txt`.

```
$ python3 glob_question.py

dir/file*.txt
dir/file1.txt
dir/file2.txt
dir/file?.txt
dir/file[.txt
dir/filea.txt
dir/fileb.txt
```

Character Ranges

Use a character range (`[a-z]`) instead of a question mark to match one of several characters. This example finds all of the files with a digit in the name before the extension.

```
# glob_charrange.py

import glob
for name in sorted(glob.glob('dir/*[0-9].*')):
    print(name)
```

The character range `[0-9]` matches any single digit. The range is ordered based on the character code for each letter/digit, and the dash indicates an unbroken range of sequential characters. The same range value could be written `[0123456789]`.

```
$ python3 glob_charrange.py

dir/file1.txt
dir/file2.txt
```

Escaping Meta-characters

Sometimes it is necessary to search for files with names containing the special meta-characters `glob` uses for its patterns. The `escape()` function builds a suitable pattern with the special characters “escaped” so they are not expanded or interpreted as special by `glob`.

```
# glob_escape.py

import glob

specials = '?*['

for char in specials:
    pattern = 'dir/*' + glob.escape(char) + '.txt'
    print('Searching for: {!r}'.format(pattern))
    for name in sorted(glob.glob(pattern)):
        print(name)
    print()
```

Each special character is escaped by building a character range containing a single entry.

```
$ python3 glob_escape.py

Searching for: 'dir/*[?].txt'
dir/file?.txt

Searching for: 'dir/*[*].txt'
dir/file*.txt

Searching for: 'dir/*[[].txt'
dir/file[.txt
```

See also

- Standard library documentation for glob
- Pattern Matching Notation – An explanation of globbing from The Open Group’s Shell Command Language specification.
- `fnmatch` – Filename matching implementation.
- Python 2 to 3 porting notes for glob

pathlib — Filesystem Paths as Objects

fnmatch — Unix-style Glob Pattern Matching

Quick Links

Example Data
Wildcards
Single Character Wildcard
Character Ranges
Escaping Meta-characters

This page was last updated 2016-12-28.

Navigation

pathlib — Filesystem Paths as Objects
fnmatch — Unix-style Glob Pattern Matching

The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for examples for Python 2?

This Site

Module Index
Index

Other Writing

Blog
The Python Standard Library By Example