⚠ この記事は最終更新日から5年以上が経過しています。

@amowwee

投稿日 2017年10月30日　更新日 2017年10月30日

Pythonでフォルダ内のファイルリストを取得する

Python

3行で

- 手軽にやるなら `glob.glob`
- サブディレクトリまで走査するなら、python 3.4以前なら `os.walk` 、python 3.5以降なら `glob.glob`
- python 3.4以降で、その後のファイル操作まで考えるなら、 `pathlib` がお勧め

前置き

大量のファイルを一括にリネームする必要があったので、調べたことを忘備録がわりに書いておきます。
あるディレクトリに入っているファイルの一覧を取得します。応用として、特定の条件にマッチするファイルのみ抽出したり、サブディレクトリ内部のファイルまで再帰的に取得することもできます。なお、この例では以下のようなディレクトリ構成になっているとします。

```
/test_dir
└── dir_A
    ├── hoge.txt
    ├── huga.txt
    ├── nyaaan.JPG
    └── dir_B
        └── piyo.txt
```

確認環境:

- Windows7 Professional
- Python 3.5.3, 2.7.13

基本

`glob.glob` 関数でパスを取得できます。

ディレクトリ直下のファイル一覧を取得

```
import glob
glob.glob("/test_dir/dir_A/*")

# Windowsでは'\\'でもディレクトリを区切れます
glob.glob("\\test_dir\\dir_A\\*")

# 相対パスでも指定可能です
import os
os.chdir("/test_dir/dir_A")
glob.glob("./*")
```

結果1

```
# 絶対パスで指定した場合は絶対パスが返る
['/test_dir/dir_A\\dir_B',
 '/test_dir/dir_A\\hoge.txt',
 '/test_dir/dir_A\\huga.txt',
 '/test_dir/dir_A\\nyaaan.JPG']
```

取得した相対パスは os.path.abspath 関数により絶対パスへ変換できます。
ファイルの件数が多くて時間がかかる場合は、 yield でジェネレータにするとマシになるかもしれません。

関数化の例

```
def listup_files(path):
    yield [os.path.abspath(p) for p in glob.glob(path)]
```

条件指定

ワイルドカードにより検索の条件指定ができます。 glob.glob 関数では'*', '?', '[]'のワイルドカードが指定できます。それぞれの役割は以下です。'*'についてはちゃっかり上で使っていますね。

- '*': 任意の文字列とマッチする（長さ0文字～）
- '?': 任意の1文字とマッチする
- '[]': [xxx-yyy]内の範囲に含まれる文字にマッチする。例えば[0-9]とすると任意の数字1文字が指定できます。

条件指定

```
import glob
# テキストファイルのみを取得
glob.glob("/test_dir/dir_A/*.txt")

# "n"で始まるファイル,ディレクトリのみを取得
glob.glob("/test_dir/dir_A/n*")

# ディレクトリ名にワイルドカードを指定することも可能。
glob.glob("/test_dir/dir_A/*/*.txt")
```

結果2

```
['/test_dir/dir_A\\nyaaan.JPG']    # ”n”で始まる

['/test_dir/dir_A\\dir_B\\piyo.txt']    # ディレクトリ名にワイルドカード。dir_A直下のテ
```

ただし、ワイルドカードで「xxxを含まない」といった指定はできないようです。したがって、「含まない」パターンを指定したいときは

- 取得したリストに対し、さらに正規表現でパターンマッチ
- すべてのファイルのリストと、除きたいファイルのみのリストをそれぞれ用意して、set型などで差分をとる

などの工夫が必要になります。

サブディレクトリまで再帰的に含める

python3.5以降のバージョンでは、 `glob.glob` 関数で再帰的な検索ができます。
それ以前のバージョンでは、 `os.walk` 関数を使います。

再帰的な検索

```
# python 3.5以降
glob.glob("/test_dir/**.txt", recursive=True)

# python3.4以前
found = []
for root, dirs, files in os.walk("/test_dir/"):
    for filename in files:
        found.append(os.path.join(root, filename))    # ファイルのみ再帰でいい場合はここ
    for dirname in dirs:
        found.append(os.path.join(root, dirname))    # サブディレクトリまでリストに含め
print(found)
```

結果3

```
# glob.globとos.walk どちらも同じ
```

```
 '/test_dir\\dir_A\\dir_B',
 '/test_dir\\dir_A\\dir_B\\piyo.txt',
 '/test_dir\\dir_A\\hoge.txt',
 '/test_dir\\dir_A\\huga.txt',
 '/test_dir\\dir_A\\nyaaan.JPG']
```

pathlib.Pathを使う

python 3.4以降では pathlib モジュールが標準ライブラリに追加されました。
このモジュールで定義されている Path クラスでもファイルの検索ができます。
Pathクラスはファイルパスに関する操作APIが充実していて、パスの文字列をそのまま扱うよりも使いやすいです。python 3.4以降の環境であれば、上記の方法よりもこちらをお勧めします。

Pathクラスを使う

```
from pathlib import Path

# Pathオブジェクトを生成
p = Path("/test_dir/dir_A/")

# dir_A直下のファイルとディレクトリを取得
# Path.glob(pattern)はジェネレータを返す。結果を明示するためlist化しているが、普段は不要
list(p.glob("*"))

# ファイル名の条件指定
list(p.glob("*.txt"))

# 再帰的な検索
list(p.glob("**/*"))
```

結果4

```
# dir_A直下のファイルとディレクトリ
[WindowsPath('/test_dir/dir_A/dir_B'),
 WindowsPath('/test_dir/dir_A/hoge.txt'),
 WindowsPath('/test_dir/dir_A/huga.txt'),
```

```
# 条件指定
[WindowsPath('/test_dir/dir_A/hoge.txt'),
 WindowsPath('/test_dir/dir_A/huga.txt')]

# 再帰的
[WindowsPath('/test_dir/dir_A/dir_B'),
 WindowsPath('/test_dir/dir_A/hoge.txt'),
 WindowsPath('/test_dir/dir_A/huga.txt'),
 WindowsPath('/test_dir/dir_A/nyaaan.JPG'),
 WindowsPath('/test_dir/dir_A/dir_B/piyo.txt')]
```

`WindowsPath` は `Path` のサブクラスです。MacやLinux系の環境では結果が変わるはずです。

`open` などpython標準で入っている関数には、文字列の代わりにPathオブジェクトをそのまま渡せます。
一方で、外部ライブラリのなかにはPathを受け付けないものもあります。軽く調べたところ、 `numpy.loadtxt(Path_obj)` や `pandas.read_csv(Path_obj)` は正常にファイルを開けましたが、 `cv2.imread(Path_obj)` はエラーになりました。
Pathオブジェクトを通常の文字列に変換するには、 `Path.as_posix` 関数を使います。あるいは、 `str(Path)` のようにキャストしてもよいです。

Pathオブジェクトからファイル読み込み

```
from pathlib import Path
import numpy as np
import pandas as pd
import cv2

# numpyはPathオブジェクトを受け付ける
csv = Path("./sample.csv")
np.loadtxt(csv, delimiter=",")

# pandasもPathオブジェクトを受け付ける
pd.read_csv(csv)

# OpenCVはエラーになる
jpg = Path("/test_dir/dir_A/nyaaan.JPG")
cv2.imread(jpg)
```

```
# 文字列として渡せばOK
cv2.imread(jpg.as_posix())
```

結果5

```
# numpy -> 正常にファイルを読み込む（略）

# pandas -> 正常にファイルを読み込む（略）

# opencv(Path_obj)
TypeError                                 Traceback (most recent call last)
<ipython-input-4-54c7c3460c88> in <module>()
----> 1 cv2.imread(jpg)

TypeError: bad argument type for built-in operation

# opencv(Path_obj.as_posix()) -> 正常にファイルを読み込む（略）
```

私信

qiitaへの投稿は初めてです。内容やレイアウトなどのツッコミをいただけると助かります。どうぞご遠慮なさらずに。

参考資料

Python 3.6.3ドキュメント 16.1. os

python 3.6.3ドキュメント 11.1 pathlib

Pythonで再帰的にファイル・ディレクトリを探して出力する

Python3.4以降ならos.pathはさっさと捨ててpathlibを使うべき

@amowwee

フォロー

コメント

この記事にコメントはありません。

あなたもコメントしてみませんか :)

新規登録

すでにアカウントを持っている方はログイン