Pythonでのファイルの読み書き（入出力）について説明する。ファイルの中身を文字列やリストとして取得したり、ファイル作成、上書き、追記したりする方法など。

open(), withによるファイル読み書き（入出力）

エンコード指定: 引数encoding

テキストファイルの読み込み

読み込み用でファイルオープン: 引数mode=r

ファイル全体を文字列として読み込み: read()

ファイル全体をリストとして読み込み: readlines()

ファイルを一行ずつ読み込み: readline()

テキストファイルの書き込み（新規作成・上書き）

書き込み用でファイルオープン: 引数mode=w

文字列を書き込み: write()

リストを書き込み: writelines()

空のファイルを作成: pass文

ファイルが存在しない場合のみ書き込み（新規作成のみ）

新規作成専用でファイルオープン: 引数mode=x

ファイルの存在を確認してからオープン

テキストファイルの書き込み（追記・挿入）

末尾に追記

先頭、途中に挿入

バイナリファイルの読み書き

ここではパスの文字列を使って処理をする組み込み関数open()について説明するが、Python3.4以降ではパスをオブジェクトとして操作できるpathlibモジュールを使うこともできる。

組み込み関数open()と同等のopen()メソッドのほか、テキストファイルやバイナリファイル全体を読み書きする便利なメソッドもあるので、一行ずつ処理したり追記したりする必要がなければpathlibを使ってみるのをおすすめする。

関連記事: Python, pathlibでファイルの作成・open・読み書き・削除

また、Web上のファイルをダウンロード・保存したい場合は以下の記事を参照。

関連記事: PythonでWeb上の画像などのファイルをダウンロード（個別・一括）

スポンサーリンク

open, withによるファイル読み書き（入出力）

読み込み・書き込みいずれの場合も組み込み関数open()でファイルを開く。

2. 組み込み関数 open() — Python 3.6.5 ドキュメント

第一引数にパスの文字列を指定するとそのパスが示すファイルオブジェクトが開かれる。読み書きの指定またはテキストファイルかバイナリファイルかの指定などは引数modeを使う（後述）。

テキストファイルの場合はio.TextIOWrapper型として読み込まれる。

path = 'data/src/test.txt'

f = open(path)

print(type(f))

# <class '\_io.TextIOWrapper'>

f.close()

source: file\_io\_with\_open.py

上の例のようにopen()したファイルオブジェクトはclose()メソッドでクローズする必要がある。

withブロックを使うとブロックの終了時に自動的にクローズされる。閉じ忘れがなくなるので便利。

with open(path) as f:

print(type(f))

# <class '\_io.TextIOWrapper'>

source: file\_io\_with\_open.py

with open() as xxx:のxxxには任意の名前を使用できる。open()でオープンしたファイルオブジェクトにxxxという名前をつけてブロック内で使用するイメージ。fが使われることが多いが、他の名前でも問題ない。

エンコーディング指定: 引数encoding

テキストファイル読み込み時のデコード、テキストファイル書き込み時のエンコードで使われるエンコーディングはopen()の引数encodingで指定する。

テキストファイルの読み込み

読み込み用でファイルオープン: mode='r'

open()の引数modeを'r'とすると読み込み用としてファイルが開かれる。引数modeのデフォルト値は'r'なので、省略してもOK。

mode='r'では第一引数に存在しないパスを指定するとエラー（FileNotFoundError例外）となる。

# with open('data/src/test\_error.txt') as f:

# print(type(f))

# FileNotFoundError: [Errno 2] No such file or directory: 'data/src/test\_error.txt'

source: file\_io\_with\_open.py

ファイル全体を文字列として読み込み: read()

ファイルオブジェクトのread()メソッドを使うと、開いたファイル全体を文字列として取得できる。

with open(path) as f:

s = f.read()

print(type(s))

print(s)

# <class 'str'>

# line 1

# line 2

# line 3

source: file\_io\_with\_open.py

ファイルオブジェクトはwithブロックの終了時にクローズされるが、当然ながら、代入した変数はwithブロックの外（ファイルオブジェクトをクローズした後）でも使える。

with open(path) as f:

s = f.read()

print(s)

# line 1

# line 2

# line 3

source: file\_io\_with\_open.py

ファイル全体をリストとして読み込み: readlines()

ファイルオブジェクトのreadlines()メソッドを使うと、開いたファイル全体を行ごとに分割したリストとして取得できる。改行コード\nを含んだ文字列が要素となる。

with open(path) as f:

l = f.readlines()

print(type(l))

print(l)

# <class 'list'>

# ['line 1\n', 'line 2\n', 'line 3']

source: file\_io\_with\_open.py

改行コードを削除したい場合はリスト内包表記を使って各要素でstrip()を呼べばOK。

関連記事: Pythonリスト内包表記の使い方

with open(path) as f:

l\_strip = [s.strip() for s in f.readlines()]

print(l\_strip)

# ['line 1', 'line 2', 'line 3']

source: file\_io\_with\_open.py

リストのインデックス指定[]をすることで、任意の行の文字列（改行コード込み）を取得できる。

with open(path) as f:

l = f.readlines()

print(l[1])

# line 2

#

source: file\_io\_with\_open.py

ファイルを一行ずつ読み込み: readline()

ファイルオブジェクトをそのままfor文で回すとファイルの先頭から一行ずつ文字列として取得できる。

with open(path) as f:

for s\_line in f:

print(s\_line)

# line 1

#

# line 2

#

# line 3

#

source: file\_io\_with\_open.py

ファイルオブジェクトのreadline()メソッドでも一行ずつ取得できる。

with open(path) as f:

s\_line = f.readline()

print(s\_line)

# line 1

#

source: file\_io\_with\_open.py

readline()メソッドで読んでいくと、最後（EOF）には空文字列''が返される。

with open(path) as f:

while True:

s\_line = f.readline()

print(s\_line)

if not s\_line:

break

# line 1

#

# line 2

#

# line 3

#

source: file\_io\_with\_open.py

いずれの場合も行末の改行コードを含む文字列が取得できる。

テキストファイルの書き込み（新規作成・上書き）

書き込み用でファイルオープン: mode='w'

open()の引数modeを'w'とすると書き込み用としてファイルが開かれる。

ファイルが存在しなければ新規作成、存在していれば上書き（既存の内容は削除）で保存される。追記・挿入については後述。

path\_w = 'data/src/test\_w.txt'

s = 'New file'

with open(path\_w, mode='w') as f:

f.write(s)

with open(path\_w) as f:

print(f.read())

# New file

source: file\_io\_with\_open.py

存在しないファイルは新規作成されるが、その直上までのディレクトリ（フォルダ）は存在していなければエラー（FileNotFoundError例外）となる。

# with open('data/src/new\_dir/test\_w.txt', mode='w') as f:

# f.write(s)

# FileNotFoundError: [Errno 2] No such file or directory: 'data/src/new\_dir/test\_w.txt'

source: file\_io\_with\_open.py

新しいディレクトリにファイルを新規作成したい場合は以下の記事を参照。

関連記事: Pythonで新しいディレクトリにファイルを作成・保存

文字列を書き込み: write()

ファイルオブジェクトのwrite()メソッドを使うと文字列を書き込むことができる。

s = 'New file'

with open(path\_w, mode='w') as f:

f.write(s)

with open(path\_w) as f:

print(f.read())

# New file

source: file\_io\_with\_open.py

改行を含む文字列もそのまま書き込まれる。

s = 'New line 1\nNew line 2\nNew line 3'

with open(path\_w, mode='w') as f:

f.write(s)

with open(path\_w) as f:

print(f.read())

# New line 1

# New line 2

# New line 3

source: file\_io\_with\_open.py

文字列の改行については以下の記事も参照。

関連記事: Pythonで改行を含む文字列の出力、連結、分割、削除、置換

write()の引数には文字列のみ指定可能。それ以外の型だとエラー（TypeError例外）となる。

整数型intや浮動小数点型floatなどの数値や他の型を書き込みたい場合はstr()で文字列に変換してからwrite()に渡す。

リストを書き込み: writelines()

ファイルオブジェクトのwritelines()メソッドを使うとリストを書き込むことができる。

l = ['One', 'Two', 'Three']

with open(path\_w, mode='w') as f:

f.writelines(l)

with open(path\_w) as f:

print(f.read())

# OneTwoThree

source: file\_io\_with\_open.py

writelines()では改行コードは挿入されず、要素がそのまま連結されて書き込まれる。

リストの要素ごとに改行して書き込みたい場合は、改行コードとjoin()メソッドで改行込みの文字列を作成し、write()メソッドで書き込む。

with open(path\_w, mode='w') as f:

f.write('\n'.join(l))

with open(path\_w) as f:

print(f.read())

# One

# Two

# Three

source: file\_io\_with\_open.py

writelines()の引数に指定できるのは文字列を要素とするリストのみ。

整数型intや浮動小数点型floatなどの数値や他の型を要素とするリストを書き込みたい場合は文字列のリストに変換する必要がある。以下の記事を参照。

関連記事: Pythonで文字列のリスト（配列）と数値のリストを相互に変換

空のファイルを作成: pass文

空のファイルを作成したい場合は書き込みモード（mode='w'）のopen()でファイルを新規作成した上で中身を何も書き込まない。

文法上、withブロックの中には何かを記述する必要があるので、何もしない文であるpass文を使う。

with open('temp/empty.txt', 'w'):

pass

source: pass\_with\_open.py

pass文の詳細については以下の記事を参照。

関連記事: Pythonのpass文の意味と使い方

ファイルが存在しない場合のみ書き込み（新規作成のみ）

上述のように、引数mode='w'では、ファイルが存在しなければ新規作成、存在していれば上書き（既存の内容は削除）となり、誤って既存のファイルを指定しまった場合は内容が上書きされてしまう。

ファイルが存在しない場合のみ書き込み、つまり、新規作成のみで上書きは禁止するには以下の二つの方法がある。

新規作成専用でファイルオープン: mode='x'

open()の引数modeを'x'とすると、ファイルが存在する場合にはエラー（FileExistsError）となり、ファイルが存在しない場合のみファイルが書き込み用として開かれ保存（新規作成）される。

バージョン3.3で追加された。

# with open(path\_w, mode='x') as f:

# f.write(s)

# FileExistsError: [Errno 17] File exists: 'data/src/test\_w.txt'

source: file\_io\_with\_open.py

try, exceptで例外処理をすれば、ファイルが存在しない場合は新規作成、ファイルが存在している場合は何もしないという処理が可能。

try:

with open(path\_w, mode='x') as f:

f.write(s)

except FileExistsError:

pass

source: file\_io\_with\_open.py

ファイルの存在を確認してからオープン

標準ライブラリのosモジュールのos.path.isfile()でファイルの存在確認ができる。

これを利用してファイルが存在しない場合のみopen()で書き込み処理を行うことが可能。

import os

if not os.path.isfile(path\_w):

with open(path\_w, mode='w') as f:

f.write(s)

source: file\_io\_with\_open.py

ファイルの存在確認についての詳細は以下の記事を参照。

関連記事: Pythonでファイル、ディレクトリ（フォルダ）の存在確認

テキストファイルの追記・挿入

末尾に追記: mode='a'

open()の引数modeを'a'とすると追記モードでファイルが開かれる。

write(), writelines()を使うと既存のファイルの末尾に追記できる。

with open(path\_w, mode='a') as f:

f.write('Four')

with open(path\_w) as f:

print(f.read())

# One

# Two

# ThreeFour

source: file\_io\_with\_open.py

最終行を追加したい場合は改行コードも含めて追記する。

with open(path\_w, mode='a') as f:

f.write('\nFour')

with open(path\_w) as f:

print(f.read())

# One

# Two

# ThreeFour

# Four

source: file\_io\_with\_open.py

なお、存在しないファイルに対してはmode='w'と同じくファイルが新規作成される。

先頭、途中に挿入

mode='r+'

open()の引数modeを'r+'とすると、読み書き用でファイルが開かれる。

write(), writelines()を使うと既存のファイルの先頭に書き込まれるが、挿入ではなく上書きとなる。

with open(path\_w, mode='r+') as f:

f.write('123456')

with open(path\_w) as f:

print(f.read())

# 123456o

# ThreeFour

# Four

source: file\_io\_with\_open.py

ファイルオブジェクトのメソッドseek()で位置を移動して書き込むことも可能だが、行単位ではなく文字単位で指定する必要がある。

with open(path\_w, mode='r+') as f:

f.seek(3)

f.write('-')

with open(path\_w) as f:

print(f.read())

# 123-56o

# ThreeFour

# Four

source: file\_io\_with\_open.py

readlines(), insert()を使う

ファイルオブジェクトのメソッドreadlines()、リストオブジェクトのメソッドinsert()を使うと、先頭の行や途中の行に新たな文字列を挿入できる。

readlines()でファイルをリストとして読み込んで、insert()で要素を挿入した後にwritelines()で書き込む。

insert()の第一引数で挿入する行数を指定できる。

with open(path\_w) as f:

l = f.readlines()

l.insert(0, 'FIRST\n')

with open(path\_w, mode='w') as f:

f.writelines(l)

with open(path\_w) as f:

print(f.read())

# FIRST

# 123-56o

# ThreeFour

# Four

source: file\_io\_with\_open.py

リストへの要素の追加・挿入については以下の記事を参照。

関連記事: Pythonでリスト（配列）に要素を追加するappend, extend, insert

リストの要素を削除すると行の削除になる。

関連記事: Pythonでリスト（配列）の要素を削除するclear, pop, remove, del

バイナリファイルの読み書き

これまで説明したmodeの末尾にbをつけるとバイナリファイルとしての読み書きとなる。

例えばmode='rb'はバイナリファイルの読み込み、moode='ab'はバイナリファイルの末尾に追記となる。

テキストファイルと同様、バイナリファイルでもファイルオブジェクトのメソッドとしてread(), readline(), readlines(), write(), writelinesが使える。

バイナリファイルの編集のほか、Webサイトからダウンロードした画像を保存する際などにも使う。