@IT
atmarkIT

解決！ Python
CSVファイル編

かわさきしんじ，Deep Insider 編集部［著］

［解決！Python］CSV ファイルの読み書きまとめ

csv モジュール／ NumPy ／ pandas を使って、CSV ファイルを読み書きする方法を 1 ページにまとめて紹介。

（2021 年 09 月 28 日）

csv モジュールを使った読み込み（csv.reader オブジェクト／ csv.DictReader クラス）

```
# csv モジュールを使って CSV ファイルから 1 行ずつ読み込む
import csv

filename = 'populationdata.csv'
with open(filename, encoding='utf8', newline='') as f:
    csvreader = csv.reader(f)
    for row in csvreader:
        print(row)
# 出力結果：
#['年', '地域コード', '地域', '総人口']
#['1920 年', '00000', '全国', '55963053']
# …… 省略 ……
#['2010 年', '00000', '全国', '128057352']
#['2020 年', '00000', '全国', '126226568']

# タブ区切りの文字を読み込む
filename = 'tabdelimiteddata.tsv'
with open(filename, encoding='utf8', newline='') as f:
    csvreader = csv.reader(f, delimiter='¥t')
    for row in csvreader:
        print(row)  # 出力結果は省略

# CSV ファイルの内容を 1 つのリストにまとめる
filename = 'populationdata.csv'
with open(filename, encoding='utf8', newline='') as f:
    csvreader = csv.reader(f)
    content = [row for row in csvreader]  # 各年のデータを要素とするリスト
    #content = []
```

```
    #for row in csvreader:
    #    content.append(row)

print(content)
# 出力結果：
#[['年', '地域コード', '地域', '総人口'], ['1920年', '00000', '全国',
# '55963053'], …… , ['2010年', '00000', '全国', '128057352'], ['2020年',
# '00000', '全国', '126226568']]

# 特定の列のデータ型を変換する
with open(filename, encoding='utf8', newline='') as f:
    csvreader = csv.reader(f)
    header = next(csvreader)  # 見出し行は別扱い
    content = [[row[0], row[1], row[2], int(row[3])] for row in csvreader]

content.insert(0, header)  # 最後にリストの先頭に見出し行を挿入
print(content)
# 出力結果：
#[['年', '地域コード', '地域', '総人口'], ['1920年', '00000', '全国',
# 55963053], …… , ['2010年', '00000', '全国', 128057352], ['2020年',
# '00000', '全国', 126226568]]

# 数値フィールド以外はシングルクオートで囲まれていることを指示して
# 数値フィールドの値を全て浮動小数点数値に自動的に変換
filename = 'sample.csv'
with open(filename, encoding='utf8', newline='') as f:
    csvreader = csv.reader(f, quotechar="'", quoting=csv.QUOTE_NONNUMERIC)
    for row in csvreader:
        print(row)
# 出力結果
# ['年', '地域コード', '地域', '総人口']
# ['1920年', '00000', '全国', 55963053.0]
# …… 省略 ……
# ['2010年', '00000', '全国', 128057352.0]
# ['2020年', '00000', '全国', 126226568.0]
```

```
# CSV ファイルの内容から辞書形式のデータを作成
with open(filename, encoding='utf8', newline='') as f:
    csvreader = csv.DictReader(f)
    content = [row for row in csvreader]

print(content[0])
# 出力結果:
#{'年': '1920 年', '地域コード': '00000', '地域': '全国', '総人口': '55963053'}
```

上のサンプルコードは以下に示す CSV ファイルからの読み込みを行う例である。

```
年 , 地域コード , 地域 , 総人口
1920 年 ,00000, 全国 ,55963053
1930 年 ,00000, 全国 ,64450005
1940 年 ,00000, 全国 ,73075071
1950 年 ,00000, 全国 ,84114574
1960 年 ,00000, 全国 ,94301623
1970 年 ,00000, 全国 ,104665171
1980 年 ,00000, 全国 ,117060396
1990 年 ,00000, 全国 ,123611167
2000 年 ,00000, 全国 ,126925843
2010 年 ,00000, 全国 ,128057352
2020 年 ,00000, 全国 ,126226568
```

出典 : 政府統計の総合窓口（e-Stat）
上記サイトの時系列表から得られる総人口のデータを加工して利用しています。

csv モジュールを使った CSV ファイルからの読み込みについては「CSV ファイルから読み込みを行うには（csv モジュール編）」を参照のこと。

csv モジュールを使った書き込み（csv.writer オブジェクトの writerow ／ writerows メソッド、csv.DictWriter クラス）

```
import csv

# サンプルデータとテキストファイルの内容を出力する関数の準備
header = ['name', 'age', 'tel']
isshiki = ['一色', 25, 'xxxx-yyyy']
endo = ['遠藤', 45, 'mmmm-nnnnn']
kawasaki = ['かわさき', 80, 'zzzz-aaaa']
mylist = [isshiki, endo, kawasaki]

for row in mylist:
    print(row)
# 出力結果：
#['一色', 25, 'xxxx-yyyy']
#['遠藤', 45, 'mmmm-nnnnn']
#['かわさき', 80, 'zzzz-aaaa']

from pathlib import Path
def print_lines():
    print(Path('test.csv').read_text())

# csv モジュールを使って 1 行の内容を CSV ファイルに書き込み
with open('test.csv', 'w', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(isshiki)

print_lines()  # 一色 ,25,xxxx-yyyy

# csv モジュールを使って複数行の内容を CSV ファイルに書き込み
with open('test.csv', 'w', newline='') as f:
    writer = csv.writer(f)
    writer.writerows(mylist)
```


→目次に戻る

```
print_lines()
# 出力結果：
# 一色 ,25,xxxx-yyyy
# 遠藤 ,45,mmmm-nnnnn
# かわさき ,80,zzzz-aaaa

# ヘッダーを行頭に書き込む
with open('test.csv', 'w', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(header)
    writer.writerows(mylist)

print_lines()
# 出力結果：
#name,age,tel
# 一色 ,25,xxxx-yyyy
# 遠藤 ,45,mmmm-nnnnn
# かわさき ,80,zzzz-aaaa

# 区切り文字をカンマからタブに変更する
with open('test.tsv', 'w', newline='') as f:
    writer = csv.writer(f, delimiter='¥t')
    writer.writerow(isshiki)

repr(Path('test.tsv').read_text())  # "' 一色 ¥¥t25¥¥txxxx-yyyy¥¥n'"

# 全てのフィールドを引用符で囲む（デフォルトはダブルクオート）
with open('test.csv', 'w', newline='') as f:
    writer = csv.writer(f, quoting=csv.QUOTE_ALL)
    writer.writerow(isshiki)

print_lines()  # " 一色 ","25","xxxx-yyyy"

# 数値以外のフィールドをシングルクオートで囲む
with open('test.csv', 'w', newline='') as f:
```

```
    writer = csv.writer(f, quoting=csv.QUOTE_NONNUMERIC, quotechar="'")
    writer.writerow(isshiki)

print_lines()  # '一色',25,'xxxx-yyyy'

# 辞書の要素をCSVファイルに書き込む
mydict_list = [dict(zip(header, row)) for row in mylist]
for items in mydict_list:
    print(items)
# 出力結果:
#{'name': '一色', 'age': 25, 'tel': 'xxxx-yyyy'}
#{'name': '遠藤', 'age': 45, 'tel': 'mmmm-nnnnn'}
#{'name': 'かわさき', 'age': 80, 'tel': 'zzzz-aaaa'}

with open('test.csv', 'w', newline='') as f:
    writer = csv.DictWriter(f, fieldnames=header)
    writer.writeheader()
    writer.writerows(mydict_list)

print_lines()
# 出力結果:
#name,age,tel
#一色,25,xxxx-yyyy
#遠藤,45,mmmm-nnnnn
#かわさき,80,zzzz-aaaa
```

csv モジュールを使った CSV ファイルへの書き込みについては「CSV ファイルに書き込みを行うには（csv モジュール編）」を参照のこと。

NumPy を使った読み込み（numpy.loadtxt ／ np.genfromtxt 関数）

```
import numpy as np
from pathlib import Path

# numpy.loadtxt 関数

# 読み込む CSV ファイルの内容を確認
filename = 'nptest0.csv'
print(Path(filename).read_text())
#0 1 2  # 数値が空白文字で区切られている
#3 4 5
#6 7 8

# loadtxt 関数の基本的な使い方
myarray = np.loadtxt(filename)  # デフォルトでは空白文字が区切り文字
print(myarray)  # デフォルトでは読み込んだ値は浮動小数点数値となる
#[[0. 1. 2.]
# [3. 4. 5.]
# [6. 7. 8.]]

# データ型を指定
myarray = np.loadtxt(filename, dtype=int)  # 全てのフィールドが整数と指定
print(myarray)
#[[0 1 2]
# [3 4 5]
# [6 7 8]]

# 読み込む CSV ファイルの内容を確認
filename = 'nptest1.csv'
print(Path(filename).read_text())
#0,1,2  # 数値がカンマで区切られている
#3,4,5
#6,7,8
```


→目次に戻る

```
# 区切り文字を指定
myarray = np.loadtxt(filename, delimiter=',')  # 区切り文字としてカンマを指定
print(myarray)
#[[0. 1. 2.]
# [3. 4. 5.]
# [6. 7. 8.]]

myarray = np.loadtxt(filename)  # ValueError：デフォルトは空白文字が区切り文字

# 読み込む CSV ファイルの内容を確認
filename = 'nptest2.csv'
print(Path(filename).read_text())
#col1 col2 col3  # ヘッダーあり。数値が空白文字で区切られている
#0 1 2
#3 4 5
#6 7 8

# 先頭行を読み飛ばす
myarray = np.loadtxt(filename, skiprows=1)  # 先頭の 1 行を読み飛ばす
print(myarray)
#[[0. 1. 2.]
# [3. 4. 5.]
# [6. 7. 8.]]

# 読み込む CSV ファイルの内容を確認
filename = 'nptest3.csv'
with open(filename, encoding='utf8') as f:
    print(f.read())
# 一色 25 170  # 文字列／数値／数値が空白文字で区切られている
# かわさき 80 168

# 読み込む列を指定する
myarray = np.loadtxt(filename, usecols=[1, 2], encoding='utf8')
print(myarray)
#[[ 25. 170.]
```


→目次に戻る

```
# [ 80. 168.]]

# 列ごとに内容を取り出す（転置）
ages, heights = np.loadtxt(filename, unpack=True, usecols=[1, 2], encoding='
utf8')
print(ages)  # [25. 80.]
print(heights)  # [170. 168.]

# numpy.genfromtxt 関数

# 読み込む CSV ファイルの内容を確認
filename = 'nptest4.csv'
with open(filename, encoding='utf8') as f:
    print(f.read())
#name age height  # ヘッダーあり。文字列／数値／数値が空白文字で区切られている
# 一色 25 170
# かわさき 80 168

# 基本的な読み込み
myarray = np.genfromtxt(filename, encoding='utf8')
print(myarray)
#[[ nan  nan  nan]  # 数値以外は nan とされている
# [ nan  25. 170.]
# [ nan  80. 168.]]
print(myarray.dtype)  # float64

# 各フィールドの値からデータ型を判定
myarray = np.genfromtxt(filename, encoding='utf8', dtype=None)
print(myarray)
#[['name' 'age' 'height']
# ['一色' '25' '170']
# ['かわさき' '80' '168']]
print(myarray.dtype)  # <U6（全てが 6 文字以下の文字列と判定された）

# 1 行目をヘッダーとして扱い、各列のデータ型を明示的に指定
```


→目次に戻る

```
myarray = np.genfromtxt(filename, names=True, dtype='U4,f,f', encoding='utf8')
print(myarray)  # [('一色', 25., 170.) ('かわさき', 80., 168.)]
print(myarray.dtype)  # [('name', '<U4'), ('age', '<f4'), ('height', '<f4')]

# 読み込む CSV ファイルの内容を確認
filename = 'nptest5.csv'
with open(filename, encoding='utf8') as f:
    print(f.read())
#0,,2  # 各行に欠損している値がある
#,4,5
#6,,8

# 欠損値を nan で埋める
myarray = np.genfromtxt(filename, delimiter=',', encoding='utf8')
print(myarray)
#[[ 0. nan  2.]  # 欠損している部分の値は nan になる
# [nan  4.  5.]
# [ 6. nan  8.]]

myarray = np.loadtxt(filename, delimiter=',', encoding='utf8')  # ValueError

# 欠損値を埋める値を指定する
myarray = np.genfromtxt(filename, delimiter=',', filling_values=-1, encoding='
utf8')
print(myarray)
#[[ 0. -1.  2.]
# [-1.  4.  5.]
# [ 6. -1.  8.]]
```

NumPy を使った CSV ファイルからの読み込みについては「CSV ファイルから読み込みを行うには（NumPy 編）」を参照のこと。

NumPy を使った書き込み（numpy.savetxt 関数）

```
import numpy as np
from pathlib import Path

x = np.random.randn(2, 3)  # 以下の値は一例
print(x)
#[[-2.45567984  1.33310634  0.59013369]
# [ 0.25731195  0.78458477 -0.64572527]]

# 基本的な使い方
np.savetxt('test.csv', x)
print(Path('test.csv').read_text())
#-2.455679835942072398e+00 1.333106339746787494e+00 5.9013369228478218 53e-
01
#2.573119457585911207e-01 7.845847696453233100e-01 -6.457252711716952032e-
01

# 区切り文字を変更する
np.savetxt('test.csv', x, delimiter=',')  # 区切り文字をカンマ「,」に
print(Path('test.csv').read_text())
#-2.455679835942072398e+00,1.333106339746787494e+00,5.901336922847821853e-01
#2.573119457585911207e-01,7.845847696453233100e-01,-6.457252711716952032e-01

# 書き出すフォーマットの指定

# 指数表記
np.savetxt('test.csv', x, fmt='%.8e')  # 小数点以下の精度を 8 桁に
print(Path('test.csv').read_text())
#-2.45567984e+00 1.33310634e+00 5.90133692e-01
#2.57311946e-01 7.84584770e-01 -6.45725271e-01

np.savetxt('test.csv', x, fmt='%18.8e')  # 最小の出力幅を指定
```


→目次に戻る

```
print(Path('test.csv').read_text())
#   -2.45567984e+00     1.33310634e+00     5.90133692e-01
#    2.57311946e-01     7.84584770e-01    -6.45725271e-01

# 浮動小数点数値（指数表記をしない）
np.savetxt('test.csv', x, fmt='%12.8f')  # 精度の後に「f」を指定
print(Path('test.csv').read_text())
# -2.45567984   1.33310634   0.59013369
#  0.25731195   0.78458477  -0.64572527

# 左寄せ
np.savetxt('test.csv', x, fmt='%-18.8e')  #「-」で左寄せを指定
print(Path('test.csv').read_text())
#-2.45567984e+00    1.33310634e+00     5.90133692e-01
#2.57311946e-01     7.84584770e-01     -6.45725271e-01

# 符号を常に付加
np.savetxt('test.csv', x, fmt='%+18.8e')  #「+」で符号を常に付加
print(Path('test.csv').read_text())
#   -2.45567984e+00    +1.33310634e+00    +5.90133692e-01
#   +2.57311946e-01    +7.84584770e-01    -6.45725271e-01

# 0埋め
np.savetxt('test.csv', x, fmt='%018.8e')  #「0」で0埋めを指定
print(Path('test.csv').read_text())
#-0002.45567984e+00 00001.33310634e+00 00005.90133692e-01
#00002.57311946e-01 00007.84584770e-01 -0006.45725271e-01

# 整数値
nums = np.array([[111, 222, 333],
                 [444, 555, 666]])

np.savetxt('test.csv', nums, fmt='%d')
print(Path('test.csv').read_text())
#111 222 333
```


→目次に戻る

```
#444 555 666

np.savetxt('test.csv', nums, fmt='%.5d')  # 0埋め
print(Path('test.csv').read_text())
#00111 00222 00333
#00444 00555 00666

np.savetxt('test.csv', nums, fmt='%6d')  # 出力される最小の文字数を指定
print(Path('test.csv').read_text())
#    111    222    333
#    444    555    666

np.savetxt('test.csv', nums, fmt='%6.4d')  # 数字の最小文字数を指定
print(Path('test.csv').read_text())
#  0111   0222   0333
#  0444   0555   0666

# 改行文字を変更する
np.savetxt('test.csv', x, newline='¥n¥n')
print(Path('test.csv').read_text())
#-2.455679835942072398e+00 1.333106339746787494e+00 5.901336922847821853e-
01
#
#2.573119457585911207e-01 7.845847696453233100e-01 -6.457252711716952032e-
01
#

# ヘッダーを付加する
np.savetxt('test.csv', x, header='col1 col2 col3')
print(Path('test.csv').read_text())
## col1 col2 col3
#-2.455679835942072398e+00 1.333106339746787494e+00 5.901336922847821853e-
01
#2.573119457585911207e-01 7.845847696453233100e-01 -6.457252711716952032e-
01
```


→目次に戻る

```
# フッターを付加する
np.savetxt('test.csv', x, footer='generated: 2021/08/27')
print(Path('test.csv').read_text())
#-2.455679835942072398e+00 1.333106339746787494e+00 5.901336922847821853e-
01
#2.573119457585911207e-01 7.845847696453233100e-01 -6.457252711716952032e-
01
## generated: 2021/08/27
```

NumPy を使った CSV ファイルへの書き込みについては「CSV ファイルに書き込みを行うには（NumPy 編）」を参照のこと。

pandas を使った読み込み（pandas.read_csv 関数）

```
import pandas as pd
from pathlib import Path

filepath = 'pdtest0.csv'
print(Path(filepath).read_text())
#0.0,1.1,2.2
#3.3,4.4,5.5
#6.6,7.7,8.8

df = pd.read_csv(filepath)
print(df)
#   0.0  1.1  2.2
#0  3.3  4.4  5.5
#1  6.6  7.7  8.8

# ヘッダー行がないことを指定
df = pd.read_csv(filepath, header=None)
print(df)
#     0    1    2
#0  0.0  1.1  2.2
#1  3.3  4.4  5.5
```

```
#2  6.6  7.7  8.8

# 列名を指定
names = ['col0', 'col1', 'col2']
df = pd.read_csv(filepath, names=names)
print(df)
#   col0  col1  col2
#0   0.0   1.1   2.2
#1   3.3   4.4   5.5
#2   6.6   7.7   8.8

# ヘッダー行がある場合
filepath = 'pdtest1.csv'
print(Path(filepath).read_text())
#col0,col1,col2
#0.0,1.1,2.2
#3.3,4.4,5.5
#6.6,7.7,8.8

df = pd.read_csv(filepath)
print(df)
#   col0  col1  col2  # ヘッダー行から列名を推測してくれる
#0   0.0   1.1   2.2
#1   3.3   4.4   5.5
#2   6.6   7.7   8.8

# 列名をヘッダー行から推測せずに、明示的に指定する
names = ['foo', 'bar', 'baz']
df = pd.read_csv(filepath, names=names, header=0)
print(df)
#   foo  bar  baz
#0  0.0  1.1  2.2
#1  3.3  4.4  5.5
#2  6.6  7.7  8.8
```

```
# 区切り文字の指定
filepath = 'pdtest2.csv'
print(Path(filepath).read_text())
#col0 col1 col2  # 空白文字で区切っている
#0.0 1.1 2.2
#3.3 4.4 5.5
#6.6 7.7 8.8

df = pd.read_csv(filepath, sep=' ')
print(df)
#  col0 col1 col2
#0    0.0 1.1 2.2
#1    3.3 4.4 5.5
#2    6.6 7.7 8.8

# 読み込む列の指定
filepath = 'pdtest1.csv'
df = pd.read_csv(filepath, usecols=[0, 2])
print(df)
#   col0  col2
#0   0.0   2.2
#1   3.3   5.5
#2   6.6   8.8

df = pd.read_csv(filepath, usecols=['col0', 'col2'])
print(df)  # 出力は省略

df = pd.read_csv(filepath, usecols=lambda x: x in ['col0', 'col2'])
print(df)  # 出力は省略

# 行ラベル（インデックス）となる列の指定
filepath = 'pdtest3.csv'
print(Path(filepath).read_text())
#,col1,col2,col3
#row0,0.0,1.1,2.2
```


→目次に戻る

```
#row1,3.3,4.4,5.5
#row2,6.6,7.7,8.8

df = pd.read_csv(filepath, index_col=0)
print(df)
#      col1  col2  col3
#IDX
#row0   0.0   1.1   2.2
#row1   3.3   4.4   5.5
#row2   6.6   7.7   8.8

df = pd.read_csv(filepath, index_col='IDX')
print(df)  # 出力は省略

# データ型の指定
filepath = 'pdtest4.csv'
print(Path(filepath).read_text())
#area,tel,value
#tokyo,0312345678,1.0
#kanagawa,045678901,2.0
#chiba,043210987,3.0

df = pd.read_csv(filepath)
print(df)
#       area        tel  value  # 電話番号が整数値になっている
#0     tokyo  312345678    1.0
#1  kanagawa   45678901    2.0
#2     chiba   43210987    3.0

df = pd.read_csv(filepath, dtype=str)  # 全てのデータの型を str に
print(df)
#       area         tel value
#0     tokyo  0312345678   1.0
#1  kanagawa   045678901   2.0
#2     chiba   043210987   3.0
```


→目次に戻る

```
df = pd.read_csv(filepath, dtype={0: str, 1: str, 2: float})
print(df)  # 出力は省略

# 日付のパース
filepath = 'pdtest5.csv'
print(Path(filepath).read_text())
#date,value0,value1
#2021/09/07,1.0,2.0
#2021/09/08,3.0,4.0
#2021/09/09,5.0,5.0

df = pd.read_csv(filepath, parse_dates=True, index_col=0)
print(df)
#            value0  value1
#date
#2021-09-07     1.0     2.0
#2021-09-08     3.0     4.0
#2021-09-09     5.0     5.0

df = pd.read_csv(filepath, parse_dates=[0])
print(df)
#        date  value0  value1
#0 2021-09-07     1.0     2.0
#1 2021-09-08     3.0     4.0
#2 2021-09-09     5.0     5.0

df = pd.read_csv(filepath, parse_dates=['date'])
print(df)  # 出力は省略

filepath = 'pdtest6.csv'
print(Path(filepath).read_text())
#year,month,day,value0,value1
#2021,9,7,1.0,2.0
#2021,9,8,3.0,5.0
#2021,9,9,4.0,6.0
```


→目次に戻る

```
dates = [['year', 'month', 'day']]
df = pd.read_csv(filepath, parse_dates=dates)
print(df)
#  year_month_day  value0  value1
#0     2021-09-07     1.0     2.0
#1     2021-09-08     3.0     5.0
#2     2021-09-09     4.0     6.0

dates = {'date': ['year', 'month', 'day']}
df = pd.read_csv(filepath, parse_dates=dates)
print(df)
#        date  value0  value1
#0 2021-09-07     1.0     2.0
#1 2021-09-08     3.0     5.0
#2 2021-09-09     4.0     6.0

# 欠損値の扱い
filepath = 'pdtest7.csv'
print(Path(filepath).read_text())
# col0,col1,col2
# ,nan,1.0
# 2.0,N/A,null
# NaN,3.0,--

df = pd.read_csv(filepath)
print(df)
#   col0  col1 col2
#0   NaN   NaN  1.0
#1   2.0   NaN  NaN
#2   NaN   3.0   --  # 「--」という文字列は欠損値としては扱われていない

df = pd.read_csv(filepath, na_values=['--'])
print(df)
#   col0  col1  col2  # デフォルトの欠損値と na_values に指定した値が欠損値
#0   NaN   NaN   1.0
```


→目次に戻る

```
#1   2.0   NaN   NaN
#2   NaN   3.0   NaN

df = pd.read_csv(filepath, keep_default_na=False,  na_values=['--'])
print(df)
#  col0 col1  col2  # na_values に指定した値のみが欠損値として扱われる
#0       nan   1.0
#1  2.0  N/A  null
#2  NaN  3.0   NaN
```

pandas を使った CSV ファイルからの読み込みについては「CSV ファイルから読み込みを行うには（pandas 編）」を参照のこと。

pandas を使った書き込み（pandas.DataFrame.to_csv メソッド）

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

data = {
    'name': ['isshiki', 'endo', 'kawasaki'],
    'age': [20, 25, np.nan],
    'weight': [55.44, 66.77, 123.456]
}
df = pd.DataFrame(data)
print(df)
#       name   age   weight
#0   isshiki  20.0   55.440
#1      endo  25.0   66.770
#2  kawasaki   NaN  123.456

# 基本
fname = 'test.csv'
df.to_csv(fname)
print(Path(fname).read_text())
```

```
#,name,age,weight
#0,isshiki,20.0,55.44
#1,endo,25.0,66.77
#2,kawasaki,,123.456

# 区切り文字の変更
df.to_csv(fname, sep=' ')
print(Path(fname).read_text())
# name age weight
#0 isshiki 20.0 55.44
#1 endo 25.0 66.77
#2 kawasaki  123.456

# 欠損値の表現を指定する
df.to_csv(fname, na_rep='nan')
print(Path(fname).read_text())
#,name,age,weight
#0,isshiki,20.0,55.44
#1,endo,25.0,66.77
#2,kawasaki,nan,123.456

# 数値を文字列化する際の書式指定
df.to_csv(fname, float_format='%+08.3f')
print(Path(fname).read_text())
#,name,age,weight
#0,isshiki,+020.000,+055.440
#1,endo,+025.000,+066.770
#2,kawasaki,,+123.456

# ヘッダー行の指定
df.to_csv(fname, header=['col0', 'col1', 'col2'])
print(Path(fname).read_text())
#,col0,col1,col2
#0,isshiki,20.0,55.44
#1,endo,25.0,66.77
```

```
#2,kawasaki,,123.456

# 行インデックスの列名を指定
df.to_csv(fname, index_label='idx')
print(Path(fname).read_text())
#idx,name,age,weight
#0,isshiki,20.0,55.44
#1,endo,25.0,66.77
#2,kawasaki,,123.456

# 行インデックスを出力しない
df.to_csv(fname, index=False)
print(Path(fname).read_text())
#name,age,weight
#isshiki,20.0,55.44
#endo,25.0,66.77
#kawasaki,,123.456

# 書き出す列の指定
df.to_csv(fname, columns=['name', 'weight'])
print(Path(fname).read_text())
#,name,weight
#0,isshiki,55.44
#1,endo,66.77
#2,kawasaki,123.456

# クオートの指定
import csv
df.to_csv(fname, quoting=csv.QUOTE_ALL)
print(Path(fname).read_text())
#"","name","age","weight"
#"0","isshiki","20.0","55.44"
#"1","endo","25.0","66.77"
#"2","kawasaki","","123.456"
```

```
df.to_csv(fname, quoting=csv.QUOTE_NONNUMERIC, quotechar="'")
print(Path(fname).read_text())
#'','name','age','weight'
#0,'isshiki',20.0,55.44
#1,'endo',25.0,66.77
#2,'kawasaki','',123.456

# フィールドを囲むのに使う引用符自体がフィールドに含まれている場合の処理
data = {
    "name": ["chak'n", "and pop"],
    "value": [100, 120]
}
df = pd.DataFrame(data)
print(df)
#       name  value
#0    chak'n    100
#1  and pop    120

df.to_csv(fname, quoting=csv.QUOTE_NONNUMERIC, quotechar="'")
print(Path(fname).read_text())
#'','name','value'
#0,'chak''n',100
#1,'and pop',120

df.to_csv(fname, sep=' ', quoting=csv.QUOTE_NONE, escapechar='¥¥')
print(Path(fname).read_text())
# name value
#0 chak'n 100
#1 and¥ pop 120
```

pandas を使った CSV ファイルへの書き込みについては「CSV ファイルに書き込みを行うには（pandas 編）」を参照のこと。

［解決！Python］CSV ファイルから読み込みを行うには（csv モジュール編）

pandas や Numpy を使わずに、Python に標準で付属する csv モジュールを使って、CSV ファイルから読み込みを行う方法を紹介する。

（2023 年 10 月 04 日）

* 本稿は 2021 年 08 月 03 日に公開された記事を Python 3.12.0 で動作確認したものです（確認日：2023 年 10 月 04 日）。

```
# csv モジュールを使って CSV ファイルから 1 行ずつ読み込む
import csv

filename = 'populationdata.csv'
with open(filename, encoding='utf8', newline='') as f:
    csvreader = csv.reader(f)
    for row in csvreader:
        print(row)
# 出力結果：
#['年', '地域コード', '地域', '総人口']
#['1920 年', '00000', '全国', '55963053']
# …… 省略 ……
#['2010 年', '00000', '全国', '128057352']
#['2020 年', '00000', '全国', '126226568']

# タブ区切りの文字を読み込む
filename = 'tabdelimiteddata.tsv'
with open(filename, encoding='utf8', newline='') as f:
    csvreader = csv.reader(f, delimiter='¥t')
    for row in csvreader:
        print(row)  # 出力結果は省略

# CSV ファイルの内容を 1 つのリストにまとめる
filename = 'populationdata.csv'
with open(filename, encoding='utf8', newline='') as f:
    csvreader = csv.reader(f)
    content = [row for row in csvreader]  # 各年のデータを要素とするリスト
```


→目次に戻る

```
    #content = []
    #for row in csvreader:
    #    content.append(row)

print(content)
# 出力結果:
#[['年', '地域コード', '地域', '総人口'], ['1920年', '00000', '全国',
# '55963053'], …… , ['2010年', '00000', '全国', '128057352'], ['2020年',
# '00000', '全国', '126226568']]

# 特定の列のデータ型を変換する
with open(filename, encoding='utf8', newline='') as f:
    csvreader = csv.reader(f)
    header = next(csvreader)  # 見出し行は別扱い
    content = [[row[0], row[1], row[2], int(row[3])] for row in csvreader]

content.insert(0, header)  # 最後にリストの先頭に見出し行を挿入
print(content)
# 出力結果:
#[['年', '地域コード', '地域', '総人口'], ['1920年', '00000', '全国',
# 55963053], …… , ['2010年', '00000', '全国', 128057352], ['2020年',
# '00000', '全国', 126226568]]

# 数値フィールド以外はシングルクオートで囲まれていることを指示して
# 数値フィールドの値を全て浮動小数点数値に自動的に変換
filename = 'sample.csv'
with open(filename, encoding='utf8', newline='') as f:
    csvreader = csv.reader(f, quotechar="'", quoting=csv.QUOTE_NONNUMERIC)
    for row in csvreader:
        print(row)
# 出力結果
# ['年', '地域コード', '地域', '総人口']
# ['1920年', '00000', '全国', 55963053.0]
# …… 省略 ……
# ['2010年', '00000', '全国', 128057352.0]
```


→目次に戻る

```
# ['2020年', '00000', '全国', 126226568.0]

# CSVファイルの内容から辞書形式のデータを作成
with open(filename, encoding='utf8', newline='') as f:
    csvreader = csv.DictReader(f)
    content = [row for row in csvreader]

print(content[0])
# 出力結果:
#{'年': '1920年', '地域コード': '00000', '地域': '全国', '総人口': '55963053'}
```

本稿では、サンプルのCSVファイルの内容として以下を利用する。1行目は見出し行であり、各行には各年／地域コード／地域／総人口の4つのデータが含まれている。

```
年,地域コード,地域,総人口
1920年,00000,全国,55963053
1930年,00000,全国,64450005
1940年,00000,全国,73075071
1950年,00000,全国,84114574
1960年,00000,全国,94301623
1970年,00000,全国,104665171
1980年,00000,全国,117060396
1990年,00000,全国,123611167
2000年,00000,全国,126925843
2010年,00000,全国,128057352
2020年,00000,全国,126226568
```

出典：政府統計の総合窓口（e-Stat）
上記サイトの時系列表から得られる総人口のデータを加工して利用しています。

csv モジュールを使った CSV ファイルの読み込み

Python には csv モジュールが標準で添付されている。これをインポートすることで、CSV ファイルの読み込みが行える（ただし、CSV ファイルの読み込みをより柔軟な形で行うのであれば、pandas や NumPy を使った方がよいだろう）。

その基本的な手順は次の通り。

1. CSV ファイルを表すファイルオブジェクトを作成
2. そのファイルオブジェクトを、csv.reader 関数に渡して、reader オブジェクトを取得
3. reader オブジェクトは反復処理可能なので、for 文や内包表記などを使って、CSV ファイルの内容を読み込む

実際の例を以下に示す。

```
import csv

filename = 'populationdata.csv'
with open(filename, encoding='utf8', newline='') as f:
    csvreader = csv.reader(f)
    for row in csvreader:
        print(row)
# 出力結果：
#['年', '地域コード', '地域', '総人口']
#['1920年', '00000', '全国', '55963053']
# …… 省略 ……
#['2010年', '00000', '全国', '128057352']
#['2020年', '00000', '全国', '126226568']
```

この例では、open 関数で本稿冒頭に示したサンプルの CSV ファイルをオープンし、そのファイルオブジェクトを csv.reader 関数に渡して reader オブジェクトを取得している。その後は、for 文に反復可能オブジェクトとして reader オブジェクトを渡して、CSV から 1 行ずつ読み込みを行い、各行の内容を表示しているだけだ。

なお、open 関数の newline 引数には空文字列を渡しているが、これは行末コードの変換を行わないことを意味している。これは、CSV ファイルのフィールド要素に改行が含まれている場合に、それらを適切に解釈するために推奨されている。

カンマ「,」ではなく、タブ文字でフィールドが区切られている場合には、csv.reader 関数の delimiter 引数に '¥t' を渡せば、タブ区切りのファイルからの読み込みも可能だ。

```
filename = 'tabdelimiteddata.tsv'
with open(filename, encoding='utf8', newline='') as f:
    csvreader = csv.reader(f, delimiter='¥t')
    for row in csvreader:
        print(row)  # 出力結果は省略
```

内包表記を使って、CSV ファイルの内容を 1 つのリストに読み込む

CSV から 1 行ずつ読み込みを行って、逐次的に処理を行うのではなく、CSV ファイルの内容を 1 つのオブジェクトとして読み込んでしまうのであれば、リスト内包表記を使ってリストのリストを作成するのが簡単だ。

以下に例を示す。

```
filename = 'populationdata.csv'
with open(filename, encoding='utf8', newline='') as f:
    csvreader = csv.reader(f)
    content = [row for row in csvreader]  # 各年のデータを要素とするリスト
    #content = []
    #for row in csvreader:
    #    content.append(row)

print(content)
# 出力結果:
#[['年', '地域コード', '地域', '総人口'], ['1920年', '00000', '全国',
# '55963053'], …… , ['2010年', '00000', '全国', '128057352'], ['2020年',
# '00000', '全国', '126226568']]
```

この例では、open 関数を呼び出して得たファイルオブジェクトから reader オブジェクトを作成したら、内包表記を使って、各行の内容（年／地域コード／地域／総人口）を要素とするリストを作成している（コメントアウトしてあるのは、同様な処理を for 文で書いたものだ）。

フィールドの型を変換する

CSV ファイルから読み込んだ内容は通常、全て文字列値となる。そのため、整数値や浮動小数点数値に変換する必要があれば、以下のように内包表記の内部で型変換をするとよい。

例えば、サンプルの CSV ファイルでは各行には年／地域コード／地域／総人口の 4 つのデータが含まれている。最初の 3 つは文字列が適切だが、最後の総人口は整数値となっていた方が扱いやすいだろう。そこで、以下のようにして最後のフィールドだけ整数値に変換を行える。

```
filename = 'populationdata.csv'
with open(filename, encoding='utf8', newline='') as f:
    csvreader = csv.reader(f)
    header = next(csvreader)  # 見出し行は特別扱い
    content = [[row[0], row[1], row[2], int(row[3])] for row in csvreader]

content.insert(0, header)  # 最後にリストの先頭に見出し行を挿入
print(content)
# 出力結果：
#[['年', '地域コード', '地域', '総人口'], ['1920年', '00000', '全国',
# 55963053], …… , ['2010年', '00000', '全国', 128057352], ['2020年',
# '00000', '全国', 126226568]]
```

この例では、最後のフィールドだけ int 関数で総人口を整数値に変換している。このとき見出し行には数値データが含まれていないので、特別扱いするのを忘れないようにしよう。

また、CSV ファイルのフォーマットによるが、数値以外のデータは全て引用符で囲まれているといった場合には、quoting キーワード引数に csv.QUOTE_NONNUMERIC を指定することで、引用符に囲まれていないデータを全て浮動小数点数値に変換させることも可能だ。以下に例を示す。

```
filename = 'sample.csv'
with open(filename, encoding='utf8', newline='') as f:
    csvreader = csv.reader(f, quotechar="'", quoting=csv.QUOTE_NONNUMERIC)
    for row in csvreader:
        print(row)
# 出力結果
# ['年', '地域コード', '地域', '総人口']
# ['1920年', '00000', '全国', 55963053.0]
# …… 省略 ……
# ['2010年', '00000', '全国', 128057352.0]
# ['2020年', '00000', '全国', 126226568.0]
```

ここでは quoting 引数に quoting=csv.QUOTE_NONNUMERIC を指定して数値データ以外のフィールドは全て引用符で囲まれていることと、quotechar 引数に "'" を指定してそれらのフィールドはシングルクオートで囲まれていることを指示している。最後のフィールドの値が浮動小数点数値になっている点に注目されたい。

CSV ファイルの内容から辞書形式のデータを作成

各行について、列名とその列のデータの組から成るデータを作成したいのであれば、csv.DictReader クラスを使用する。csv.reader 関数と同じように、csv.DictReader クラスのコンストラクタ呼び出しにファイルオブジェクトを渡し、csv.DictReader オブジェクトを得たら、それを使って反復処理を行うだけだ。

以下に例を示す。

```
filename = 'populationdata.csv'
with open(filename, encoding='utf8', newline='') as f:
    csvreader = csv.DictReader(f)
    content = [row for row in csvreader]

print(content[0])
# 出力結果:
#{'年': '1920年', '地域コード': '00000', '地域': '全国', '総人口': '55963053'}
```

ここで使用しているサンプルの CSV ファイルでは 1 行目が見出し行となっている。そのため、csv.DictReader 呼び出しにはファイルオブジェクトだけを渡している。このときには、1 行目がフィールド名として解釈される。最初の要素の出力を見ると、「{' 年 ': '1920 年 ', ' 地域コード ': '00000', ' 地域 ': ' 全国 ', ' 総人口 ': '55963053'}」のように辞書になっていることが確認できる。

CSV ファイルに見出し行がないときには、以下のように、fieldnames 引数に見出しを渡す。

```
filename = 'noheader.csv'
with open(filename, encoding='utf8', newline='') as f:
    fieldnames = ['年','地域コード','地域','総人口']
    csvreader = csv.DictReader(f, fieldnames=fieldnames)
    for row in csvreader:
        print(row)
```

［解決！Python］CSV ファイルに書き込みを行うには（csv モジュール編）

pandas や NumPy を使わずに、Python に標準で付属する csv モジュールを使って、CSV ファイルに書き込みを行う方法を紹介する。

（2021 年 08 月 17 日）

```
import csv

# サンプルデータとテキストファイルの内容を出力する関数の準備
header = ['name', 'age', 'tel']
isshiki = ['一色', 25, 'xxxx-yyyy']
endo = ['遠藤', 45, 'mmmm-nnnnn']
kawasaki = ['かわさき', 80, 'zzzz-aaaa']
mylist = [isshiki, endo, kawasaki]

for row in mylist:
    print(row)
# 出力結果：
#['一色', 25, 'xxxx-yyyy']
#['遠藤', 45, 'mmmm-nnnnn']
#['かわさき', 80, 'zzzz-aaaa']

from pathlib import Path
def print_lines():
    print(Path('test.csv').read_text())

# csv モジュールを使って 1 行の内容を CSV ファイルに書き込み
with open('test.csv', 'w', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(isshiki)

print_lines()  # 一色 ,25,xxxx-yyyy

# csv モジュールを使って複数行の内容を CSV ファイルに書き込み
with open('test.csv', 'w', newline='') as f:
```

```
    writer = csv.writer(f)
    writer.writerows(mylist)

print_lines()
# 出力結果：
# 一色 ,25,xxxx-yyyy
# 遠藤 ,45,mmmm-nnnnn
# かわさき ,80,zzzz-aaaa

# ヘッダーを行頭に書き込む
with open('test.csv', 'w', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(header)
    writer.writerows(mylist)

print_lines()
# 出力結果：
#name,age,tel
# 一色 ,25,xxxx-yyyy
# 遠藤 ,45,mmmm-nnnnn
# かわさき ,80,zzzz-aaaa

# 区切り文字をカンマからタブに変更する
with open('test.tsv', 'w', newline='') as f:
    writer = csv.writer(f, delimiter='¥t')
    writer.writerow(isshiki)

repr(Path('test.tsv').read_text())  # "'一色¥¥t25¥¥txxxx-yyyy¥¥n'"

# 全てのフィールドを引用符で囲む（デフォルトはダブルクオート）
with open('test.csv', 'w', newline='') as f:
    writer = csv.writer(f, quoting=csv.QUOTE_ALL)
    writer.writerow(isshiki)

print_lines()  # "一色","25","xxxx-yyyy"
```


→目次に戻る

```
# 数値以外のフィールドをシングルクオートで囲む
with open('test.csv', 'w', newline='') as f:
    writer = csv.writer(f, quoting=csv.QUOTE_NONNUMERIC, quotechar="'")
    writer.writerow(isshiki)

print_lines()  # '一色',25,'xxxx-yyyy'

# 辞書の要素をCSVファイルに書き込む
mydict_list = [dict(zip(header, row)) for row in mylist]
for items in mydict_list:
    print(items)
# 出力結果:
#{'name': '一色', 'age': 25, 'tel': 'xxxx-yyyy'}
#{'name': '遠藤', 'age': 45, 'tel': 'mmmm-nnnnn'}
#{'name': 'かわさき', 'age': 80, 'tel': 'zzzz-aaaa'}

with open('test.csv', 'w', newline='') as f:
    writer = csv.DictWriter(f, fieldnames=header)
    writer.writeheader()
    writer.writerows(mydict_list)

print_lines()
# 出力結果:
#name,age,tel
#一色,25,xxxx-yyyy
#遠藤,45,mmmm-nnnnn
#かわさき,80,zzzz-aaaa
```


→目次に戻る

csv モジュールを使った CSV ファイルの書き込み

Python には csv モジュールが標準で添付されている。これをインポートすることで、リストなどの内容の CSV ファイルへの書き込みが行える（ただし、CSV 形式のデータの読み書きをより柔軟な形で行うのであれば、pandas や NumPy を使った方がよいだろう）。

その基本的な手順は次の通り。

1. CSV ファイルを表すファイルオブジェクトを作成
2. そのファイルオブジェクトを、csv.writer 関数に渡して、writer オブジェクトを取得
3. writerow メソッドを使って 1 行分のデータを、あるいは writerows メソッドを使って複数行分のデータを CSV ファイルに書き込む

ここでは、次のようなデータ（と、書き出したテキストファイルの内容を表示する関数）を使って、その方法を見ていくことにする。

```
header = ['name', 'age', 'tel']
isshiki = ['一色', 25, 'xxxx-yyyy']
endo = ['遠藤', 45, 'mmmm-nnnnn']
kawasaki = ['かわさき', 80, 'zzzz-aaaa']
mylist = [isshiki, endo, kawasaki]

for row in mylist:
    print(row)
# 出力結果：
#['一色', 25, 'xxxx-yyyy']
#['遠藤', 45, 'mmmm-nnnnn']
#['かわさき', 80, 'zzzz-aaaa']

from pathlib import Path
def print_lines():
    print(Path('test.csv').read_text())
```

上で紹介した手順で 1 行分のデータを CSV ファイルに書き込む簡単な例を以下に示す。

```
import csv

with open('test.csv', 'w', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(isshiki)

print_lines()  # 一色 ,25,xxxx-yyyy
```

この例では、test.csv ファイルを書き込みモードでオープンし、そのファイルオブジェクトを csv.writer 関数に渡して、writer オブジェクトを取得している。次に、writerow メソッドを使って、リストの要素を CSV ファイルへ書き込んでいる。writer オブジェクトは、与えられたデータを区切り文字で区切った文字列に変換して、ファイルに書き込みを行う。

open 関数の newline 引数には空文字列を渡しているが、これは行末コードの変換を行わないことを意味している。これは CSV ファイルのフィールド要素に改行文字が含まれている場合に、それらを適切に解釈できるようにするためのものだ（csv.writer 関数の説明でそうすることが推奨されている）。

print_lines 関数の出力結果を見ると、リストの要素がカンマ区切りで分割されて書き込まれていることが分かる。

複数行のデータをまとめて CSV ファイルに書き込むには writerows メソッドを使用する。以下に例を示す。

```
with open('test.csv', 'w', newline='') as f:
    writer = csv.writer(f)
    writer.writerows(mylist)

print_lines()
# 出力結果：
# 一色 ,25,xxxx-yyyy
# 遠藤 ,45,mmmm-nnnnn
# かわさき ,80,zzzz-aaaa
```

ここでは、writerows メソッドにリストを要素とするリスト（リストのリスト）を渡している。これにより、名前／年齢／電話番号を格納しているリストが 1 行分のデータとして書き込まれ、全体としては 3 行分のデータの書き込みが行われている。

CSV ファイルにヘッダー行が必要であれば、データを書き込む前に別途ヘッダーを書き込めばよい。以下に例を示す。

```
with open('test.csv', 'w', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(header)
    writer.writerows(mylist)

print_lines()
# 出力結果：
#name,age,tel
# 一色 ,25,xxxx-yyyy
# 遠藤 ,45,mmmm-nnnnn
# かわさき ,80,zzzz-aaaa
```

辞書の内容を CSV ファイルに書き込む場合には、DictWriter オブジェクトの作成時に fieldnames 引数にヘッダー情報を与えた上で、writeheader メソッドを使ってもよい（後述）。

区切り文字をカンマからタブに変更する

上で見たように、csv.writer 関数によって得られる writer オブジェクトはデフォルトでリストなどの反復可能オブジェクトの要素をカンマで区切って、ファイルに書き出す。区切り文字を変更したいのであれば、csv.writer 関数の呼び出し時に delimiter 引数に区切り文字を渡してやる。例えば、以下はタブを区切り文字にする例だ。

```
with open('test.tsv', 'w', newline='') as f:
    writer = csv.writer(f, delimiter='¥t')
    writer.writerow(isshiki)

repr(Path('test.tsv').read_text())  # "'一色¥¥t25¥¥txxxx-yyyy¥¥n'"
```

テキストを読み込んで、repr 関数でその表現を確認すると、カンマの区切りにタブ文字が使われているのが確認できた。タブ区切りのデータを読み込む際には、もちろん区切り文字を指定する必要がある点には注意。

```
with open('test.tsv', newline='') as f:
    reader = csv.reader(f, delimiter='¥t')
    for row in reader:
        print(row)  # ['一色', '25', 'xxxx-yyyy']
```

フィールドを引用符で囲む

CSV の書き込みを行う際には、各フィールドを引用符で囲むこともできる。デフォルトでは、あるフィールドの値として区切り文字が含まれている場合などに引用符で囲まれるようになっている。

```
tmp = ['1, 2, 3', '4, 5, 6']
with open('test.csv', 'w', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(tmp)

print_lines()  # "1, 2, 3","4, 5, 6"
```

例えば、このコードでは文字列リストの要素に（デフォルトの区切り文字である）カンマが含まれている。そのため、文字列要素をダブルクオートで囲んだものをカンマで区切って書き出している。

csv.writer 関数の呼び出し時には、quoting 引数に引用符の使い方（引用符で囲む／囲まない／特定のフィールドだけを囲むなど）を指定できる。指定できるのは以下の値だ。

- csv.QUOTE_ALL：全てのフィールドを引用符で囲む
- csv.QUOTE_MINIMAL：フィールドの値に特別な文字（区切り文字、引用符、改行文字など）が含まれている場合にのみ引用符で囲む（デフォルト値）
- csv.QUOTE_NONNUMERIC：数値以外のフィールドは全て引用符で囲む
- csv.QUOTE_NONE：全てのフィールドを引用符で囲まない

以下に例を示す。

```
with open('test.csv', 'w', newline='') as f:
    writer = csv.writer(f, quoting=csv.QUOTE_ALL)
    writer.writerow(isshiki)

print_lines()  # "一色","25","xxxx-yyyy"
```

この例では、全てのフィールドを引用符で囲むように指定しているので、テキストファイルの内容を見ると、全てが（デフォルトの引用符である）ダブルクオートで囲まれている。

引用符を変更するには、csv.writer 関数の呼び出し時に quotechar 引数に引用符として使用する文字を指定する。

以下の例では、引用符をシングルクオートに、また、数値以外のフィールドのみを引用符で囲むように指定している。

```
with open('test.csv', 'w', newline='') as f:
    writer = csv.writer(f, quoting=csv.QUOTE_NONNUMERIC, quotechar="'")
    writer.writerow(isshiki)

print_lines()  # '一色',25,'xxxx-yyyy'
```

引用符のデフォルトのダブルクオートから変更した場合には、そのファイルの内容を読み込むために csv.reader 関数で reader オブジェクトを作成する際にも、quotechar 引数に引用符を指定するのを忘れないようにしよう（コードは省略）。

```
with open('test.csv', newline='') as f:
    reader = csv.reader(f, quotechar="'")
    for row in reader:
        print(row)
```

フィールドに引用符が含まれている場合、デフォルトでは引用符が二重化される。

```
sample_data = ['"foo', 'bar', 'baz']
with open('test.csv', 'w', newline='') as f:
    writer = csv.writer(f, quoting=csv.QUOTE_ALL)
    writer.writerow(sample_data)

print_lines()  # """foo","bar","baz"
```

この例ではリストの先頭要素の値が「"foo」となっている。そして、quotechar 引数に引用符を指定していないので、引用符はダブルクオートだ。そのため、「"foo」の先頭にあるダブルクオートが二重化されてファイルには「""foo」として書き込まれるということだ。

引用符の二重化をしたくないときには、csv.write 関数呼び出しで doublequote 引数に False を指定した上で、escapechar 引数に引用符のエスケープに使用する文字を指定する。以下に例を示す。

```
sample_data = ['"foo', 'bar', 'baz']
with open('test.csv', 'w', newline='') as f:
    writer = csv.writer(f, quoting=csv.QUOTE_ALL,
                        doublequote=False, escapechar='¥¥')
    writer.writerow(sample_data)

print_lines()  # "¥"foo","bar","baz"
```

詳細については、Python のドキュメント「Dialect クラスと書式化パラメータ」を参照されたい。

辞書に格納されているデータを CSV ファイルに書き込み

csv モジュールでは、「キー：値」の組が集まった辞書を CSV ファイルに書き込むこともできる。これには、csv.DictWriter クラスを使う。

ここでは、以下のようにして、ヘッダーに記載されている各フィールドの名前とその値を基に辞書を作成して、csv.DictWriter クラスのインスタンスを作ってみる。

```
mydict_list = [dict(zip(header, row)) for row in mylist]
for items in mydict_list:
    print(items)
# 出力結果：
#{'name': '一色', 'age': 25, 'tel': 'xxxx-yyyy'}
#{'name': '遠藤', 'age': 45, 'tel': 'mmmm-nnnnn'}
#{'name': 'かわさき', 'age': 80, 'tel': 'zzzz-aaaa'}
```

変数 mydict_list には「'name': 名前」「'age': 年齢」「'tel': 電話番号」という 3 つの要素を含んだ辞書を要素とするリストが代入されている。

この「辞書のリスト」を使って、各フィールドの値を CSV ファイルに書き込む例を以下に示す。

```
with open('test.csv', 'w', newline='') as f:
    writer = csv.DictWriter(f, fieldnames=header)
    writer.writeheader()
    writer.writerows(mydict_list)

print_lines()
# 出力結果：
#name,age,tel
#一色,25,xxxx-yyyy
#遠藤,45,mmmm-nnnnn
#かわさき,80,zzzz-aaaa
```

csv.DictWriter オブジェクトの作成時に指定する fieldnames 引数は辞書の値がどのように CSV ファイルに書き込まれるかを指定するものだ。この例では、変数 header の値は「['name', 'age', 'tel']」であり、名前、年齢、電話番号の順に CSV に書き込まれるようになる。

ヘッダーを CSV ファイルに書き込むには、writeheader メソッドを使用する。その後は、writerows メソッドに辞書を要素とするリストを渡すだけで、辞書の要素が CSV ファイルに書き込まれる。

［解決！Python］CSV ファイルから読み込みを行うには（NumPy 編）

NumPy が提供する loadtxt 関数と genfromtxt 関数を使って、CSV ファイルなどからデータを読み込む方法を紹介する。

（2021 年 08 月 24 日）

```
import numpy as np
from pathlib import Path

# numpy.loadtxt 関数

# 読み込む CSV ファイルの内容を確認
filename = 'test0.csv'
print(Path(filename).read_text())
#0 1 2  # 数値が空白文字で区切られている
#3 4 5
#6 7 8

# loadtxt 関数の基本的な使い方
myarray = np.loadtxt(filename)  # デフォルトでは空白文字が区切り文字
print(myarray)  # デフォルトでは読み込んだ値は浮動小数点数値となる
#[[0. 1. 2.]
# [3. 4. 5.]
# [6. 7. 8.]]

# データ型を指定
myarray = np.loadtxt(filename, dtype=int)  # 全てのフィールドが整数と指定
print(myarray)
#[[0 1 2]
# [3 4 5]
# [6 7 8]]

# 読み込む CSV ファイルの内容を確認
filename = 'test1.csv'
print(Path(filename).read_text())
```


→目次に戻る

```
#0,1,2  # 数値がカンマで区切られている
#3,4,5
#6,7,8

# 区切り文字を指定
myarray = np.loadtxt(filename, delimiter=',')  # 区切り文字としてカンマを指定
print(myarray)
#[[0. 1. 2.]
# [3. 4. 5.]
# [6. 7. 8.]]

myarray = np.loadtxt(filename)  # ValueError：デフォルトは空白文字が区切り文字

# 読み込む CSV ファイルの内容を確認
filename = 'test2.csv'
print(Path(filename).read_text())
#col1 col2 col3  # ヘッダーあり。数値が空白文字で区切られている
#0 1 2
#3 4 5
#6 7 8

# 先頭行を読み飛ばす
myarray = np.loadtxt(filename, skiprows=1)  # 先頭の 1 行を読み飛ばす
print(myarray)
#[[0. 1. 2.]
# [3. 4. 5.]
# [6. 7. 8.]]

# 読み込む CSV ファイルの内容を確認
filename = 'test3.csv'
with open(filename, encoding='utf8') as f:
    print(f.read())
# 一色 25 170  # 文字列／数値／数値が空白文字で区切られている
# かわさき 80 168
```


→目次に戻る

```
# 読み込む列を指定する
myarray = np.loadtxt(filename, usecols=[1, 2], encoding='utf8')
print(myarray)
#[[ 25. 170.]
# [ 80. 168.]]

# 列ごとに内容を取り出す（転置）
ages, heights = np.loadtxt(filename, unpack=True, usecols=[1, 2], encoding='
utf8')
print(ages)  # [25. 80.]
print(heights)  # [170. 168.]

# numpy.genfromtxt 関数

# 読み込む CSV ファイルの内容を確認
filename = 'test4.csv'
with open(filename, encoding='utf8') as f:
    print(f.read())
#name age height  # ヘッダーあり。文字列／数値／数値が空白文字で区切られている
# 一色 25 170
# かわさき 80 168

# 基本的な読み込み
myarray = np.genfromtxt(filename, encoding='utf8')
print(myarray)
#[[ nan  nan  nan]  # 数値以外は nan とされている
# [ nan  25. 170.]
# [ nan  80. 168.]]
print(myarray.dtype)  # float64

# 各フィールドの値からデータ型を判定
myarray = np.genfromtxt(filename, encoding='utf8', dtype=None)
print(myarray)
#[['name' 'age' 'height']
# ['一色' '25' '170']
```


→目次に戻る

```
# ['かわさき' '80' '168']]
print(myarray.dtype)  # <U6（全てが6文字以下の文字列と判定された）

# 1行目をヘッダーとして扱い、各列のデータ型を明示的に指定
myarray = np.genfromtxt(filename, names=True, dtype='U4,f,f', encoding='utf8')
print(myarray)  # [('一色', 25., 170.) ('かわさき', 80., 168.)]
print(myarray.dtype)  # [('name', '<U4'), ('age', '<f4'), ('height', '<f4')]

# 読み込むCSVファイルの内容を確認
filename = 'test5.csv'
with open(filename, encoding='utf8') as f:
    print(f.read())
#0,,2  # 各行に欠損している値がある
#,4,5
#6,,8

# 欠損値をnanで埋める
myarray = np.genfromtxt(filename, delimiter=',', encoding='utf8')
print(myarray)
#[[ 0. nan  2.]  # 欠損している部分の値はnanになる
# [nan  4.  5.]
# [ 6. nan  8.]]

myarray = np.loadtxt(filename, delimiter=',', encoding='utf8')  # ValueError

# 欠損値を埋める値を指定する
myarray = np.genfromtxt(filename, delimiter=',', filling_values=-1, encoding='
utf8')
print(myarray)
#[[ 0. -1.  2.]
# [-1.  4.  5.]
# [ 6. -1.  8.]]
```

NumPy には loadtxt 関数と genfromtxt 関数があり、これらを使うことで CSV ファイル（あるいは任意の文字を区切り文字としてフィールドを分割する形式のファイル）から読み込みを行える。

以下では、それら 2 つの関数の基本的な使い方を紹介する。なお、本稿では CSV ファイル（または空白文字を区切り文字とするファイル）の内容を表示してから、「このときにはこんな感じで関数を呼び出す」例を示すことにする。

numpy.loadtxt 関数

numpy.loadtxt 関数（以下、loadtxt 関数）は何らかの文字を区切り文字として、「数値」が並べられているファイルの内容を読み込んで、それを NumPy の配列（numpy.ndarray）として返送する。

以下に numpy.loadtxt 関数の構文を示す。ただし、指定可能なパラメーターはこれら以外にも多数あるので、詳細については「numpy.loadtxt」を参照されたい。

```
numpy.loadtxt(fname, dtype, delimiter, skiprows, usecols, unpack)
```

上に挙げた指定可能なパラメーターはおおよそ次のような意味を持つ。また、読み込むファイルには各行に同じ数のデータが含まれている必要がある。ファイルのエンコーディングを指定するための encoding パラメーターもあるが、これについては詳しくは触れない。

- fname：読み込むファイルの名前（またはジェネレータ）。必須
- dtype：返送する配列のデータ型（CSV ファイルにはここで指定されたデータ型の値が記述されているものと見なされる）。省略可。デフォルト値は float
- delimiter：区切り文字の指定。省略可。デフォルト値は空白文字
- skiprows：先頭から読み飛ばす行数。省略可。デフォルト値は「0」（読み飛ばさない）
- usecols：読み込みを行う列の指定（先頭列が 0、次の列が 1、……、となる）。整数値、もしくは整数値を要素とするシーケンス（リストなど）。省略可。デフォルト値は None（全ての列が読み込まれる）
- unpack：True なら読み込んだ配列を転置したものが返される。省略可。デフォルト値は False（読み込んだ配列がそのまま返される）

以下は最も基本的な呼び出しの例だ。

```
filename = 'test0.csv'
print(Path(filename).read_text())
#0 1 2  # 数値が空白文字で区切られている
#3 4 5
#6 7 8

myarray = np.loadtxt(filename)  # デフォルトでは空白文字が区切り文字
print(myarray)  # デフォルトでは読み込んだ値は浮動小数点数値となる
#[[0. 1. 2.]
# [3. 4. 5.]
# [6. 7. 8.]]

print(type(myarray))  # <class 'numpy.ndarray'>
```

loadtxt 関数を呼び出す前に、読み込むファイルの内容を確認すると、各行には 3 つの数値が空白文字で区切られて並べられていることが分かる。上で述べたように、loadtxt 関数は delimiter パラメーターに区切り文字を指定しない限り、空白文字が区切り文字として扱われるので、このようにしてある。

その後、読み込むファイルの名前だけを指定して loadtxt 関数を指定している。その戻り値は上に示した通り、浮動小数点数値を 3 つ含んだ NumPy の配列（numpy.ndarray）となっている。また、「0.」「1.」のように読み込んだ値はデフォルトで浮動小数点数値となる点にも注意しよう。

データ型を指定

ファイルに格納されているデータの型を指定するには dtype パラメーターを使用する。以下の例では「dtype=int」として整数値が含まれていることを示している。

```
myarray = np.loadtxt(filename, dtype=int)  # 全てのフィールドが整数と指定
print(myarray)
#[[0 1 2]
# [3 4 5]
# [6 7 8]]
```

区切り文字を指定

次に区切り文字を指定する方法を見る。上と同様に、読み込むファイルの内容を確認すると、今度は次のようにカンマ区切りとなっている。

```
filename = 'test1.csv'
print(Path(filename).read_text())
#0,1,2  # 数値がカンマで区切られている
#3,4,5
#6,7,8
```

区切り文字を指定するには、delimiter パラメーターを使用する。以下に例を示す。

```
myarray = np.loadtxt(filename, delimiter=',')  # 区切り文字としてカンマを指定
print(myarray)
#[[0. 1. 2.]
# [3. 4. 5.]
# [6. 7. 8.]]
```

結果は先ほどと同様である。なお、カンマを区切り文字として使っているファイルに対して、delimiter パラメーターを指定せずに loadtxt 関数を呼び出すと以下のように例外が発生する。

```
myarray = np.loadtxt(filename)  # ValueError：デフォルトは空白文字が区切り文字
```

先頭行を読み飛ばす

以下のようにヘッダー行がある場合に、その行を読み飛ばすには skiprows パラメーターに読み飛ばす行数を指定すればよい。

```
filename = 'test2.csv'
print(Path(filename).read_text())
#col1 col2 col3  # ヘッダーあり。数値が空白文字で区切られている
#0 1 2
#3 4 5
#6 7 8
```

以下に例を示す。この場合は、読み飛ばす行は 1 行でよいので、「skiprows=1」としている。

```
myarray = np.loadtxt(filename, skiprows=1)  # 先頭の 1 行を読み飛ばす
print(myarray)
#[[0. 1. 2.]
# [3. 4. 5.]
# [6. 7. 8.]]
```

読み込む列を指定する

今度は次のように、文字列と数値が混在する CSV ファイルから読み込んでみる。

```
filename = 'test3.csv'
with open(filename, encoding='utf8') as f:
    print(f.read())
# 一色 25 170  # 文字列／数値／数値が空白文字で区切られている
# かわさき 80 168
```

データ型を指定せずに読み込もうとすると次のように例外が発生する。

```
myarray = np.loadtxt(filename, encoding='utf8')  # ValueError
```

しかし、データ型を str に指定してもあまり意味はないだろう。loadtxt 関数では、特定のフィールドが特定の型という指定はできず、全てのデータがある型でなければならない。

```
myarray = np.loadtxt(filename, dtype=str, encoding='utf8')
print(myarray)
#[['一色' '25' '170']
# ['かわさき' '80' '168']]
```

このようなときに、数値のみを含んだフィールドだけを読み込みたいのであれば、usecols パラメーターに読み込みたい列を指定する（先頭列が 0、次の列が 1、……のようになる）。以下は第 1 列と第 2 列の内容だけを読み込むコードだ。

```
myarray = np.loadtxt(filename, usecols=[1, 2], encoding='utf8')
print(myarray)
#[[ 25. 170.]
# [ 80. 168.]]
```

このようなときには、以下で紹介する genfromtxt 関数を使うか、以降で紹介する pandas を使うのがよいだろう。

列ごとに内容を取り出す（転置）

先頭列の内容からなる配列、次の列の内容からなる配列などがほしいときには、unpack パラメーターに True を指定するとよい。以下に例を示す。

```
ages, heights = np.loadtxt(filename, unpack=True, usecols=[1, 2], encoding='
utf8')
print(ages)  # [25. 80.]
print(heights)  # [170. 168.]
```

numpy.genfromtxt 関数

loadtxt 関数はシンプルな（全てのデータが同一の型を持つ）場合には便利に使えるかもしれないが、numpy.genfromtxt 関数（以下、genfromtxt 関数）はもっと構造的なデータを扱える。また、読み込むファイルに欠損値が含まれている場合に、その値を自動でセットすることもできる（デフォルトでは nan 値）。ただし、より高度な処理を行うのであれば、pandas を使う方がよいだろう。

以下に genfromtxt 関数の構文を示す。loadtxt 関数と同じく、パラメーターは本稿で紹介するものだけを示しているので、詳細については「numpy.genfromtxt」を参照されたい。

```
numpy.genfromtxt(fname, dtype, names, filling_values)
```

パラメーターは以下の通り。

- fname：読み込みを行うファイルの名前。必須。fname にはファイル名以外も指定可能だが、本稿では取り上げない
- dtype：返送する配列のデータ型。省略可。省略した場合のデータ型は float
- names：True の場合、CSV ファイルの先頭行から各列の名前が決定される。文字列を要素とするリスト、列名をカンマで区切った文字列を渡すことも可能。省略可。省略した場合は None を指定したものとされる（この場合は、dtype パラメーターで列名が指定されている場合には、それが使用される）
- filling_values：欠損値があった場合に、デフォルト値として使われる値を指定。省略可。省略時は None が指定されたものとして扱われる（欠損している箇所には nan がセットされる）

また、loadtxt 関数と同様に delimiter パラメーターに区切り文字を、skip_header パラメーターに skiprows パラメーターと同じく先頭から読み飛ばす行数を指定できる。encoding パラメーターで読み込むファイルのエンコーディングも指定できる。

以下にシンプルな呼び出し例を示す。

```
filename = 'test4.csv'
with open(filename, encoding='utf8') as f:
    print(f.read())
#name age height  # ヘッダーあり。文字列／数値／数値が空白文字で区切られている
#一色 25 170
#かわさき 80 168

myarray = np.genfromtxt(filename, encoding='utf8')
print(myarray)
#[[ nan  nan  nan]  # 数値以外は nan とされている
# [ nan  25. 170.]
# [ nan  80. 168.]]
print(myarray.dtype)  # float64
```

ここでは、上に示したようにヘッダーを持ち、文字列と数値が空白文字で区切られているファイルから読み込みを行っている。

genfromtxt 関数にファイル名（とエンコーディング）だけを指定した場合、配列のデータ型はデフォルトの float となる。そのため、文字列のデータについては全て nan となる。

nan とは

nan とは「Not a Number」つまり「非数」を表す値のこと。浮動小数点数値の演算で適正な値が得られなかったことを表す値であり、IEEE 754 で規定されている。Python では NaN は float 型の値であり、以下のようにして得られる。

```
nan = float('nan')
print(type(nan))  # <class 'float'>
```

nan は他の値、および自身との比較演算において常に False を返す。そのため、ある値が nan であるかどうかを調べるには、NumPy や math モジュールが提供する isnan 関数を使用する。

```
import math
import numpy as np

print(f'{math.isnan(nan)=}')  # math.isnan(nan)=True
print(f'{np.isnan(nan)=}')  # np.isnan(nan)=True
```

各列の内容からデータ型を決定する

dtype パラメーターに None を指定すると、各フィールドの内容からデータ型が決定される。

```
myarray = np.genfromtxt(filename, encoding='utf8', dtype=None)
print(myarray)
#[['name' 'age' 'height']
# ['一色' '25' '170']
# ['かわさき' '80' '168']]
print(myarray.dtype)  # <U6（全てが 6 文字以下の文字列と判定された）
```

この場合は、全列のデータ型が最大で 6 文字の文字列「<u6」と判定されている。これは恐らくヘッダー行を読み込んでいるからだ。これを読み飛ばして、自動的にデータ型を決定するコード例を以下に示す。

```
myarray = np.genfromtxt(filename, encoding='utf8', skip_header=1, dtype=None)
print(myarray)  # [('一色', 25, 170) ('かわさき', 80, 168)]
print(myarray.dtype)  # [('f0', '<U4'), ('f1', '<i8'), ('f2', '<i8')]
```

これにより、列名が自動で付き（「f0」「f1」「f2」）、それぞれのデータ型が「<U4」「<i8」「<i8」と決定されている。

あるいは、1 行目をヘッダーとして扱い、各列のデータ型を明示的に指定することもできる。

```
myarray = np.genfromtxt(filename, names=True, dtype='U4,f,f', encoding='utf8')
print(myarray)  # [('一色', 25., 170.) ('かわさき', 80., 168.)]
print(myarray.dtype)  # [('name', '<U4'), ('age', '<f4'), ('height', '<f4')]
```

ここでは「dtype='U4,f,f'」としているが、これは先頭列が「4 文字以下の文字列」、他の 2 列が浮動小数点数型であることを意味している。

欠損値の補完

次に欠損値の扱いについて簡単に見る。ここでは、以下のような CSV ファイルがあるとする。

```
filename = 'test5.csv'
with open(filename, encoding='utf8') as f:
    print(f.read())
#0,,2  # 各行に欠損している値がある
#,4,5
#6,,8
```

このようなファイルを loadtxt 関数で読もうとすると例外が発生する。

```
myarray = np.loadtxt(filename, delimiter=',', encoding='utf8')  # ValueError
```

対して、genfromtxt 関数では、以下のように欠損している箇所の値が「nan」となる。

```
myarray = np.genfromtxt(filename, delimiter=',', encoding='utf8')
print(myarray)
#[[ 0. nan  2.]  # 欠損している部分の値は nan になる
# [nan  4.  5.]
# [ 6. nan  8.]]
```

nan ではなく、何らかの値を代わりに含めておきたいのであれば、filling_values パラメーターにその値を指定する。

```
myarray = np.genfromtxt(filename, delimiter=',', filling_values=-1, encoding='
utf8')
print(myarray)
#[[ 0. -1.  2.]
# [-1.  4.  5.]
# [ 6. -1.  8.]]
```

filling_values パラメーターには辞書を指定してもよい。辞書には欠損値があったときに代替する値を列ごとに指定する。指定しなかった列で欠損値が見つかれば、そこは nan で代替される。

```
fillv = {1: -100, 2: -1000}
myarray = np.genfromtxt(filename, delimiter=',', filling_values=fillv, encoding='
utf8')
print(myarray)
#[[   0. -100.    2.]
# [  nan    4.    5.]
# [   6. -100.    8.]]
```

［解決！Python］CSV ファイルに書き込みを行うには（NumPy 編）

NumPy が提供する savetxt 関数関数を使って、CSV ファイルなどにデータを書き込む方法を紹介する。

（2021 年 08 月 31 日）

```
import numpy as np
from pathlib import Path

x = np.random.randn(2, 3)  # 以下の値は一例
print(x)
#[[-2.45567984  1.33310634  0.59013369]
# [ 0.25731195  0.78458477 -0.64572527]]

# 基本的な使い方
np.savetxt('test.csv', x)
print(Path('test.csv').read_text())
#-2.455679835942072398e+00 1.333106339746787494e+00 5.901336922847821853e-01
#2.573119457585911207e-01 7.845847696453233100e-01 -6.457252711716952032e-01

# 区切り文字を変更する
np.savetxt('test.csv', x, delimiter=',')  # 区切り文字をカンマ「,」に
print(Path('test.csv').read_text())
#-2.455679835942072398e+00,1.333106339746787494e+00,5.901336922847821853e-01
#2.573119457585911207e-01,7.845847696453233100e-01,-6.457252711716952032e-01

# 書き出すフォーマットの指定

# 指数表記
np.savetxt('test.csv', x, fmt='%.8e')  # 小数点以下の精度を 8 桁に
print(Path('test.csv').read_text())
```

```
#-2.45567984e+00 1.33310634e+00 5.90133692e-01
#2.57311946e-01 7.84584770e-01 -6.45725271e-01

np.savetxt('test.csv', x, fmt='%18.8e')  # 最小の出力幅を指定
print(Path('test.csv').read_text())
#   -2.45567984e+00     1.33310634e+00     5.90133692e-01
#    2.57311946e-01     7.84584770e-01    -6.45725271e-01

# 浮動小数点数値（指数表記をしない）
np.savetxt('test.csv', x, fmt='%12.8f')  # 精度の後に「f」を指定
print(Path('test.csv').read_text())
# -2.45567984   1.33310634   0.59013369
#  0.25731195   0.78458477  -0.64572527

# 左寄せ
np.savetxt('test.csv', x, fmt='%-18.8e')  #「-」で左寄せを指定
print(Path('test.csv').read_text())
#-2.45567984e+00    1.33310634e+00     5.90133692e-01
#2.57311946e-01     7.84584770e-01     -6.45725271e-01

# 符号を常に付加
np.savetxt('test.csv', x, fmt='%+18.8e')  #「+」で符号を常に付加
print(Path('test.csv').read_text())
#   -2.45567984e+00    +1.33310634e+00    +5.90133692e-01
#   +2.57311946e-01    +7.84584770e-01    -6.45725271e-01

# 0埋め
np.savetxt('test.csv', x, fmt='%018.8e')  #「0」で0埋めを指定
print(Path('test.csv').read_text())
#-0002.45567984e+00 00001.33310634e+00 00005.90133692e-01
#00002.57311946e-01 00007.84584770e-01 -0006.45725271e-01

# 整数値
nums = np.array([[111, 222, 333],
                 [444, 555, 666]])
```

```
np.savetxt('test.csv', nums, fmt='%d')
print(Path('test.csv').read_text())
#111 222 333
#444 555 666

np.savetxt('test.csv', nums, fmt='%.5d')  # 0埋め
print(Path('test.csv').read_text())
#00111 00222 00333
#00444 00555 00666

np.savetxt('test.csv', nums, fmt='%6d')  # 出力される最小の文字数を指定
print(Path('test.csv').read_text())
#    111    222    333
#    444    555    666

np.savetxt('test.csv', nums, fmt='%6.4d')  # 数字の最小文字数を指定
print(Path('test.csv').read_text())
#  0111   0222   0333
#  0444   0555   0666

# 改行文字を変更する
np.savetxt('test.csv', x, newline='¥n¥n')
print(Path('test.csv').read_text())
#-2.455679835942072398e+00 1.333106339746787494e+00 5.9013369228478218 53e-
01
#
#2.573119457585911207e-01 7.845847696453233100e-01 -6.457252711716952032e-
01
#

# ヘッダーを付加する
np.savetxt('test.csv', x, header='col1 col2 col3')
print(Path('test.csv').read_text())
## col1 col2 col3
#-2.455679835942072398e+00 1.333106339746787494e+00 5.9013369228478218 53e-
```


→目次に戻る

```
01
#2.573119457585911207e-01  7.845847696453233100e-01  -6.457252711716952032e-
01

# フッターを付加する
np.savetxt('test.csv', x, footer='generated: 2021/08/27')
print(Path('test.csv').read_text())
#-2.455679835942072398e+00  1.333106339746787494e+00  5.901336922847821853e-
01
#2.573119457585911207e-01  7.845847696453233100e-01  -6.457252711716952032e-
01
## generated: 2021/08/27
```

基本的な使い方

NumPy が提供する numpy.savetxt 関数（以下、savetxt 関数）を使うと、NumPy の配列、リストなどの値を、特定の文字で区切って、CSV ファイルなどに出力できる。ただし、出力可能なのは 1 次元または 2 次元の配列やリスト（array_like オブジェクト）に限る。

savetxt 関数の構文を以下に示す。

```
numpy.savetxt(fname, x, fmt='%.18e', delimiter=' ', newline='¥n', header='',
              footer='', comments='# ', encoding=None)
```

指定可能なパラメーターは次の通り。

- fname：書き込みを行うファイルの名前。必須
- x：書き込むデータ。必須
- fmt：x に格納されているデータを出力する際に使われる書式指定。省略可能。省略時は「%.18e」が指定されたものと見なされる（精度が 18 桁の指数表記）
- delimiter：区切り文字。省略可。省略時の区切り文字は空白文字となる
- newline：行末文字の指定。省略可。省略時は改行文字「'¥n'」が指定されたものと見なされる
- header：ファイルの先頭に付加されるヘッダー行の指定。省略可。省略時にはヘッダーは付加されない
- footer：ファイルの末尾に付加されるフッター行の指定。省略可。省略時にはフッターは付加されない

- comments：コメント行であることを示す文字列。この文字列に続けて、header パラメーターと footer パラメーターで指定された文字列が書き込まれる。省略可。省略時には「# 」が指定されたものとして見なされる
- encoding：ファイルのエンコーディング指定

最も基本的な使い方を以下に示す。

```
import numpy as np
from pathlib import Path

x = np.random.randn(2, 3)
print(x)
#[[-2.45567984  1.33310634  0.59013369]
# [ 0.25731195  0.78458477 -0.64572527]]

# 基本的な使い方
np.savetxt('test.csv', x)
print(Path('test.csv').read_text())
#-2.455679835942072398e+00 1.333106339746787494e+00 5.9013369228478218 53e-
01
#2.573119457585911207e-01 7.845847696453233100e-01 -6.457252711716952032e-
01
```

この例では、2 行 3 列の NumPy 配列（numpy.ndarray）を作成して（その値はランダムであるため、読者が本稿のコードを実行したときには、違う値の配列が得られる点には注意されたい）、それを savetxt 関数に渡しているだけだ。出力されたファイルの内容を見ると、小数点以下の精度が 18 桁の指数表記の値が空白文字で区切られて出力されていることが分かる。これは fmt パラメーターのデフォルト値が「%.18e」（小数点以下の精度が 18 桁の指数表記）に、delimiter のデフォルト値が「' '」（空白文字）となっているからだ。

なお、上記では NumPy 配列を扱っているが、例えば Python のリストもこの関数を使って書き出せる。

```
mylist = x.tolist()
np.savetxt('test.csv', mylist)
print(Path('test.csv').read_text())
#7.731030073194588015e-01 -1.741182180141600311e+00 9.3484189966633227 11e-02
#4.149555709304568740e-01 -1.150815368506415970e+00 -7.909651117231002171e-02
```

区切り文字を変更する

区切り文字を変更するには、delimiter パラメーターに区切り文字を指定する。以下に例を示す。

```
np.savetxt('test.csv', x, delimiter=',')  # 区切り文字をカンマ「,」に
print(Path('test.csv').read_text())
#-2.455679835942072398e+00,1.333106339746787494e+00,5.90133692284782185 3e-01
#2.573119457585911207e-01,7.845847696453233100e-01,-6.457252711716952032e-01
```

ここでは、「delimiter=','」としているので、出力ファイルの内容を見ると、各値がカンマで区切られている。

書き出すフォーマットの指定

fmt パラメーターには、各要素を出力時にどのように書式化するかを指定できる。その値は「%[flag][width][.precision]specifier」という形式で指定する。先頭の「%」と specifier に指定する値以外は省略可能だ。

- flag：書式化された値を左寄せにするか、符号を常に付加するか、0 埋めするかを指定
- width：各値が最小で何文字を使って書式化されるかを指定
- .precision：specifier に「d」または「i」を指定したときには（整数）数字を使って出力される最小の文字数を指定。specifier に「e」、「E」、「f」を指定したときには（実数）小数点以下の精度を指定、specifier に「s」を指定したときには（文字列）文字列の最大文字数を指定

- specifier：各値を整数として書式化するときには「d」または「i」を指定する。実数として書式化するときには「e」「E」（指数表記）、または「f」（小数表記）を指定する。文字列として書式化するときには「s」を指定する。specifier には他にも「g」「G」「o」「u」「x」「X」を指定できるが、本稿では取り上げない。詳細については NumPy のドキュメント「numpy.savetxt」を参照のこと

例えば、配列の要素を指数表記で CSV ファイルに書き込むには次のようにする。

```
np.savetxt('test.csv', x, fmt='%.8e')  # 小数点以下の精度を 8 桁に
print(Path('test.csv').read_text())
#-2.45567984e+00 1.33310634e+00 5.90133692e-01
#2.57311946e-01 7.84584770e-01 -6.45725271e-01
```

上の例では「%」の後には .precision として「.8」が、specifier として「e」が記述されている（flag と width は省略）。このため、小数点以下 8 桁の指数表記の数値として CSV ファイルに書き込みが行われている。

```
np.savetxt('test.csv', x, fmt='%18.8e')  # 最小の出力幅を指定
print(Path('test.csv').read_text())
#   -2.45567984e+00     1.33310634e+00     5.90133692e-01
#    2.57311946e-01     7.84584770e-01    -6.45725271e-01
```

この例では、「%」に続けて width として「18」が、他の要素については上と同じ値が指定されている。この結果、全体としては各要素が 18 文字の文字列として出力されている。このとき、符号に 1 文字（ある場合）、整数部に 1 文字、小数点に 1 文字、小数点以下に 8 文字、指数表記に 4 文字が使われて 15 文字（または 14 文字）の文字列となるので、余った部分は空白文字で埋められている。

以下は指数表記ではない小数表記を行う例だ。

```
np.savetxt('test.csv', x, fmt='%12.8f')  # 精度の後に「f」を指定
print(Path('test.csv').read_text())
# -2.45567984   1.33310634   0.59013369
#  0.25731195   0.78458477  -0.64572527
```

この例では、「%」の後に width として「12」が、.precision として「.8」が、最後に specifier として「f」が指定されている。

書式化された出力を左寄せで CSV ファイルに出力するには flag に「-」を指定する。以下に例を示す。

```
np.savetxt('test.csv', x, fmt='%-18.8e')  #「-」で左寄せを指定
print(Path('test.csv').read_text())
#-2.45567984e+00    1.33310634e+00     5.90133692e-01
#2.57311946e-01     7.84584770e-01     -6.45725271e-01
```

この例では、全体の文字数が 18 文字で、精度は 8 桁の指数表記（e）となっているものを左寄せしている。そのため、各値の後ろにパディングとしての空白文字が付加されている。

書式化の際に常に符号を付加するには、flag に「+」を指定する。以下に例を示す。

```
np.savetxt('test.csv', x, fmt='%+18.8e')  #「+」で符号を常に付加
print(Path('test.csv').read_text())
#   -2.45567984e+00    +1.33310634e+00    +5.90133692e-01
#   +2.57311946e-01    +7.84584770e-01    -6.45725271e-01
```

ここまでは全体の文字数に、書式化後の値の文字数が満たなかった場合に空白文字でパディングが行われていたが、0 埋めをするのであれば、flag に「0」を指定する。以下に例を示す。

```
np.savetxt('test.csv', x, fmt='%018.8e')  #「0」で 0 埋めを指定
print(Path('test.csv').read_text())
#-0002.45567984e+00 00001.33310634e+00 00005.90133692e-01
#00002.57311946e-01 00007.84584770e-01 -0006.45725271e-01
```

整数を要素とする配列を書き込むときには、specifier に「d」を指定する。

```
nums = np.array([[111, 222, 333],
                 [444, 555, 666]])

np.savetxt('test.csv', nums, fmt='%d')
print(Path('test.csv').read_text())
#111 222 333
#444 555 666
```

このときには、.precision は整数として初期化された文字の最小文字数を意味するようになる。

```
np.savetxt('test.csv', nums, fmt='%.5d')  # 0 埋め
print(Path('test.csv').read_text())
#00111 00222 00333
#00444 00555 00666
```

この例では、.precision として「.5」が指定されているので、最小でも 5 文字の文字列として各値が書式化されるので、余った部分は 0 埋めされている。

一方、width は書式化された値の最大の文字数を指定するものだ。以下の例では width として「6」を指定している。

```
np.savetxt('test.csv', nums, fmt='%6d')  # 出力される最小の文字数を指定
print(Path('test.csv').read_text())
#    111    222    333
#    444    555    666
```

このときには、0 埋めではなく、空白文字列を使ってパディングが行われている。以下は width に「6」を、.precision に「.4」を指定した例だ。

```
np.savetxt('test.csv', nums, fmt='%6.4d')  # 数字の最小文字数を指定
print(Path('test.csv').read_text())
#  0111   0222   0333
#  0444   0555   0666
```

このときには、全体は 6 文字だが、数字として書式化する最小の文字数が 4 なので「 0XXX」のようになっている。

最後に fmt パラメーターは各列の書式化方法を個別に指定することもできる。詳しくは説明しないが、簡単に例だけを示しておこう。

```
fmt1 = '%.4f %.6f %.2f'  # 書式指定を区切り文字で区切った文字列
np.savetxt('test.csv', x, fmt=fmt1)
print(Path('test.csv').read_text())
#-2.4557 1.333106 0.59
#0.2573 0.784585 -0.65

fmt2 = ['%.4f', '%.8f', '%.2f']  # 書式指定を要素とするリスト
np.savetxt('test.csv', x, fmt=fmt2)
print(Path('test.csv').read_text())
# 上に同じ
```

改行文字を変更する

改行文字を変更することはあまりないだろうが、newline パラメーターを使って、変更可能だ。以下に例を示す。

```
np.savetxt('test.csv', x, newline='¥n¥n')
print(Path('test.csv').read_text())
#-2.455679835942072398e+00 1.333106339746787494e+00 5.901336922847821853e-
01
#
#2.573119457585911207e-01 7.845847696453233100e-01 -6.457252711716952032e-
01
#
```

この例では、「newline='¥n¥n'」としているので、各行の後に空行ができている（numpy.loadtxt 関数でこのファイルを読み込むことは確認した）。

ヘッダー／フッターを付加する

CSV ファイルにヘッダーやフッターを付加するには、header パラメーターと footer パラメーターを指定する。以下に例を示す。

```
# ヘッダーを付加する
np.savetxt('test.csv', x, header='col1 col2 col3')
print(Path('test.csv').read_text())
## col1 col2 col3
#-2.455679835942072398e+00 1.333106339746787494e+00 5.901336922847821853e-
01
#2.573119457585911207e-01 7.845847696453233100e-01 -6.457252711716952032e-
01

# フッターを付加する
np.savetxt('test.csv', x, footer='generated: 2021/08/27')
print(Path('test.csv').read_text())
#-2.455679835942072398e+00 1.333106339746787494e+00 5.901336922847821853e-
01
#2.573119457585911207e-01 7.845847696453233100e-01 -6.457252711716952032e-
01
## generated: 2021/08/27
```

実際のヘッダーとフッターは「# 」で始まり、それに続けて header パラメーターもしくは footer パラメーターに指定した文字列が出力されている点に注意されたい。この文字列は comments パラメーターを使って変更することも可能だ（例は割愛する）。

［解決！Python］CSV ファイルから読み込みを行うには（pandas 編）

pandas が提供する read_csv 関数を使って、CSV ファイルなどからデータを読み込む方法を紹介する。

（2021 年 09 月 07 日）

```
import pandas as pd
from pathlib import Path

filepath = 'test0.csv'
print(Path(filepath).read_text())
#0.0,1.1,2.2
#3.3,4.4,5.5
#6.6,7.7,8.8

df = pd.read_csv(filepath)
print(df)
#   0.0  1.1  2.2
#0  3.3  4.4  5.5
#1  6.6  7.7  8.8

# ヘッダー行がないことを指定
df = pd.read_csv(filepath, header=None)
print(df)
#     0    1    2
#0  0.0  1.1  2.2
#1  3.3  4.4  5.5
#2  6.6  7.7  8.8

# 列名を指定
names = ['col0', 'col1', 'col2']
df = pd.read_csv(filepath, names=names)
print(df)
#   col0  col1  col2
#0   0.0   1.1   2.2
```


→目次に戻る

```
#1   3.3   4.4   5.5
#2   6.6   7.7   8.8

# ヘッダー行がある場合
filepath = 'test1.csv'
print(Path(filepath).read_text())
#col0,col1,col2
#0.0,1.1,2.2
#3.3,4.4,5.5
#6.6,7.7,8.8

df = pd.read_csv(filepath)
print(df)
#   col0  col1  col2  # ヘッダー行から列名を推測してくれる
#0   0.0   1.1   2.2
#1   3.3   4.4   5.5
#2   6.6   7.7   8.8

# 列名をヘッダー行から推測せずに、明示的に指定する
names = ['foo', 'bar', 'baz']
df = pd.read_csv(filepath, names=names, header=0)
print(df)
#   foo  bar  baz
#0  0.0  1.1  2.2
#1  3.3  4.4  5.5
#2  6.6  7.7  8.8

# 区切り文字の指定
filepath = 'test2.csv'
print(Path(filepath).read_text())
#col0 col1 col2  # 空白文字で区切っている
#0.0 1.1 2.2
#3.3 4.4 5.5
#6.6 7.7 8.8
```

```
df = pd.read_csv(filepath, sep=' ')
print(df)
#  col0 col1 col2
#0    0.0 1.1 2.2
#1    3.3 4.4 5.5
#2    6.6 7.7 8.8

# 読み込む列の指定
filepath = 'test1.csv'
df = pd.read_csv(filepath, usecols=[0, 2])
print(df)
#   col0  col2
#0   0.0   2.2
#1   3.3   5.5
#2   6.6   8.8

df = pd.read_csv(filepath, usecols=['col0', 'col2'])
print(df)  # 出力は省略

df = pd.read_csv(filepath, usecols=lambda x: x in ['col0', 'col2'])
print(df)  # 出力は省略

# 行ラベル（インデックス）となる列の指定
filepath = 'test3.csv'
print(Path(filepath).read_text())
#,col1,col2,col3
#row0,0.0,1.1,2.2
#row1,3.3,4.4,5.5
#row2,6.6,7.7,8.8

df = pd.read_csv(filepath, index_col=0)
print(df)
#      col1  col2  col3
#IDX
#row0   0.0   1.1   2.2
```

```
#row1   3.3   4.4   5.5
#row2   6.6   7.7   8.8

df = pd.read_csv(filepath, index_col='IDX')
print(df)  # 出力は省略

# データ型の指定
filepath = 'test4.csv'
print(Path(filepath).read_text())
#area,tel,value
#tokyo,0312345678,1.0
#kanagawa,045678901,2.0
#chiba,043210987,3.0

df = pd.read_csv(filepath)
print(df)
#       area        tel  value  # 電話番号が整数値になっている
#0     tokyo  312345678    1.0
#1  kanagawa   45678901    2.0
#2     chiba   43210987    3.0

df = pd.read_csv(filepath, dtype=str)  # 全てのデータの型を str に
print(df)
#       area         tel value
#0     tokyo  0312345678   1.0
#1  kanagawa   045678901   2.0
#2     chiba   043210987   3.0

df = pd.read_csv(filepath, dtype={0: str, 1: str, 2: float})
print(df)  # 出力は省略

# 日付のパース
filepath = 'test5.csv'
print(Path(filepath).read_text())
#date,value0,value1
```

```
#2021/09/07,1.0,2.0
#2021/09/08,3.0,4.0
#2021/09/09,5.0,5.0

df = pd.read_csv(filepath, parse_dates=True, index_col=0)
print(df)
#            value0  value1
#date
#2021-09-07     1.0     2.0
#2021-09-08     3.0     4.0
#2021-09-09     5.0     5.0

df = pd.read_csv(filepath, parse_dates=[0])
print(df)
#        date  value0  value1
#0 2021-09-07     1.0     2.0
#1 2021-09-08     3.0     4.0
#2 2021-09-09     5.0     5.0

df = pd.read_csv(filepath, parse_dates=['date'])
print(df)  # 出力は省略

filepath = 'test6.csv'
print(Path(filepath).read_text())
#year,month,day,value0,value1
#2021,9,7,1.0,2.0
#2021,9,8,3.0,5.0
#2021,9,9,4.0,6.0

dates = [['year', 'month', 'day']]
df = pd.read_csv(filepath, parse_dates=dates)
print(df)
#  year_month_day  value0  value1
#0     2021-09-07     1.0     2.0
#1     2021-09-08     3.0     5.0
```


→目次に戻る

```
#2     2021-09-09     4.0     6.0

dates = {'date': ['year', 'month', 'day']}
df = pd.read_csv(filepath, parse_dates=dates)
print(df)
#         date  value0  value1
#0 2021-09-07     1.0     2.0
#1 2021-09-08     3.0     5.0
#2 2021-09-09     4.0     6.0

# 欠損値の扱い
filepath = 'test7.csv'
print(Path(filepath).read_text())
# col0,col1,col2
# ,nan,1.0
# 2.0,N/A,null
# NaN,3.0,--

df = pd.read_csv(filepath)
print(df)
#   col0  col1 col2
#0   NaN   NaN  1.0
#1   2.0   NaN  NaN
#2   NaN   3.0   --  # 「--」という文字列は欠損値としては扱われていない

df = pd.read_csv(filepath, na_values=['--'])
print(df)
#   col0  col1  col2  # デフォルトの欠損値と na_values に指定した値が欠損値
#0   NaN   NaN   1.0
#1   2.0   NaN   NaN
#2   NaN   3.0   NaN

df = pd.read_csv(filepath, keep_default_na=False,  na_values=['--'])
print(df)
#  col0 col1  col2  # na_values に指定した値のみが欠損値として扱われる
```

```
#0       nan   1.0
#1  2.0  N/A  null
#2  NaN  3.0   NaN
```

基本的な使い方

pandas の read_csv 関数を使うと、CSV ファイルの内容を pandas.DataFrame オブジェクトに読み込める。読み込んだ内容は pandas が提供するさまざまな機能を使って、参照したり加工したりできる。

read_csv 関数の構文を以下に示す。なお、記述しているパラメーターは本稿で取り上げるものだけだ。詳細については pandas のドキュメント「read_csv 関数」を参照されたい

```
pandas.read_csv(filepath, sep, header, names, index_col, usecols, dtype,
                keep_default_na, na_values, parse_date)
```

これらのパラメーターについて簡単にまとめる。

- filepath：読み込む CSV ファイルの名前。必須
- sep：区切り文字の指定（delimiter パラメーターも使える。ただし、両者を同時に指定することはできない）。省略可。省略時はカンマ「,」が指定されたものとして扱われる
- header：ヘッダー行の指定。省略可。省略時は読み込んだ内容からヘッダー行と列名を推測する
- names：列名の指定。省略可
- index_col：行ラベル（行にアクセスするためのインデックス）となる列の指定。省略可。省略時は各行には整数値でアクセスする
- usecols：読み込む列の指定。省略可。省略時は全ての列の内容が読み込まれる
- dtype：データ全体または特定列のデータのデータ型を指定。省略可。省略時はデータの内容に応じて自動的にデータ型が推測される
- keep_default_na：na_values パラメーターに欠損値と見なされる値が指定された場合に、デフォルトで欠損値として見なしている値を保持するかどうかの指定。省略可。省略時は True が指定されたものとして見なされる（デフォルトの欠損値を保持する）
- na_values：欠損値として見なされる値の追加指定。省略可
- parse_dates：日付としてパースするかどうかの指定

最も基本的な使い方を以下に示す。なお、以下では読み込む CSV ファイルの内容を示した後に、read_csv 関数を呼び出して、読み込んだ値を表示することにする。

```
import pandas as pd
from pathlib import Path

filepath = 'test0.csv'
print(Path(filepath).read_text())
#0.0,1.1,2.2
#3.3,4.4,5.5
#6.6,7.7,8.8

df = pd.read_csv(filepath)
print(df)
#   0.0  1.1  2.2
#0  3.3  4.4  5.5
#1  6.6  7.7  8.8
```

この例では、read_csv 関数を使って 3 行 3 列の CSV ファイルから読み込みを行っている。ただし、先頭行（0.0,1.1,2.2）がヘッダー行と見なされてしまっている（その内容から列名が「0.0」「1.1」「2.2」と自動的に推測されてもいる）。ヘッダー行がないことを知らせるには、次のようにする。

```
df = pd.read_csv(filepath, header=None)
print(df)
#     0    1    2
#0  0.0  1.1  2.2
#1  3.3  4.4  5.5
#2  6.6  7.7  8.8
```

header=None を指定したことで、CSV ファイルの先頭行がヘッダー行として扱われなくなり、ヘッダー行から推測されていた列名の代わりに「0」「1」「2」という列名が付けられている。

列名を明示的に指定するには、names パラメーターにそれらを指定する。以下に例を示す。

```
names = ['col0', 'col1', 'col2']
df = pd.read_csv(filepath, names=names)
print(df)
#   col0  col1  col2
#0   0.0   1.1   2.2
#1   3.3   4.4   5.5
#2   6.6   7.7   8.8
```

一方、CSV ファイルにヘッダー行があるときには、既に見たように自動的にヘッダー行が認識され、列名が推測される。

```
filepath = 'test1.csv'
print(Path(filepath).read_text())
#col0,col1,col2
#0.0,1.1,2.2
#3.3,4.4,5.5
#6.6,7.7,8.8

df = pd.read_csv(filepath)
print(df)
#   col0  col1  col2  # ヘッダー行から列名を推測してくれる
#0   0.0   1.1   2.2
#1   3.3   4.4   5.5
#2   6.6   7.7   8.8
```

列名をヘッダー行の内容と異なるものにするのであれば、以下のようにヘッダー行を明示的に指定するとともに、names パラメーターに列名を与える。

```
names = ['foo', 'bar', 'baz']
df = pd.read_csv(filepath, names=names, header=0)
print(df)
#   foo  bar  baz
#0  0.0  1.1  2.2
#1  3.3  4.4  5.5
#2  6.6  7.7  8.8
```

この例では、「header=0」としてヘッダー行が先頭行であることを指定し、同時に names パラメーターに ['foo', 'bar', 'baz'] というリストを与えているので、列名がヘッダー行とは異なるものになっている。

ヘッダー行が複数ある場合には、header=[0, 1] などと指定することも可能だ（この場合は、先頭行とその次の行の値を要素とするタプルが列にアクセスするためのインデックスとなる）。

区切り文字の指定

区切り文字を指定するには、sep パラメーターまたは delimiter パラメーターにそれを指定する。以下は、空白文字を区切り文字とするファイルから読み込みを行う例だ。

```
filepath = 'test2.csv'
print(Path(filepath).read_text())
#col0 col1 col2  # 空白文字で区切っている
#0.0 1.1 2.2
#3.3 4.4 5.5
#6.6 7.7 8.8

df = pd.read_csv(filepath, sep=' ')
print(df)
#  col0 col1 col2
#0    0.0 1.1 2.2
#1    3.3 4.4 5.5
#2    6.6 7.7 8.8
```

「sep=None」「engine='python'」を指定することで、区切り文字の自動推測が可能になる。

```
df = pd.read_csv(filepath, sep=None, engine='python')
```

読み込む列の指定

元の CSV ファイルから特定の列の内容だけを読み込みたいときには、usecols パラメーターに読み込みたい列を指定する。以下に例を示す。

```
filepath = 'test1.csv'
df = pd.read_csv(filepath, usecols=[0, 2])
print(df)
#   col0  col2
#0   0.0   2.2
#1   3.3   5.5
#2   6.6   8.8
```

この例では、第 0 列と第 2 列の内容だけを読み込むように指定している。列名を usecols パラメーターに指定してもよい。

```
df = pd.read_csv(filepath, usecols=['col0', 'col2'])
print(df)  # 出力は省略
```

usecols パラメーターにラムダ式を渡すこともできる。ラムダ式を渡した場合、そのラムダ式には列名が渡され、それを使ったラムダ式の評価結果が True となる列のみが読み込まれる。

```
df = pd.read_csv(filepath, usecols=lambda x: x in ['col0', 'col2'])
print(df)  # 出力は省略
```

この例では、列名は 'col0'、'col1'、'col2' の 3 つであり、それらがラムダ式に渡される。ラムダ式では「列名 in ['col0', 'col2']」が評価されるので、第 1 列（'col1'）以外の 2 列が読み込まれる。

行ラベル（インデックス）となる列の指定

行ラベル（行にアクセスするためのインデックス）として使用する列を指定するには、index_col パラメーターにその値（整数、列名）を指定する。以下に例を示す。

```
filepath = 'test3.csv'
print(Path(filepath).read_text())
#,col1,col2,col3
#row0,0.0,1.1,2.2
#row1,3.3,4.4,5.5
#row2,6.6,7.7,8.8

df = pd.read_csv(filepath, index_col=0)
print(df)
#       col1  col2  col3
#IDX
#row0   0.0   1.1   2.2
#row1   3.3   4.4   5.5
#row2   6.6   7.7   8.8
```

あるいは列名を指定してもよい。

```
df = pd.read_csv(filepath, index_col='IDX')
print(df)  # 出力は省略
```

データ型の指定

CSV ファイルに格納されているデータ全体、または特定の列のデータ型を指定するには、dtype パラメーターを使用する。

例えば、以下のような CSV ファイルがあったとする。

```
filepath = 'test4.csv'
print(Path(filepath).read_text())
#area,tel,value
#tokyo,0312345678,1.0
#kanagawa,045678901,2.0
#chiba,043210987,3.0
```

第 0 列は地域、第 1 列は電話番号、第 2 列は何らかの値を示すデータとなっている。これを、何も指定せずに read_csv 関数で読み込むと次のようになる。

```
df = pd.read_csv(filepath)
print(df)
#       area        tel  value
#0     tokyo  312345678    1.0
#1  kanagawa   45678901    2.0
#2     chiba   43210987    3.0

df.info()
#<class 'pandas.core.frame.DataFrame'>
#RangeIndex: 3 entries, 0 to 2
#Data columns (total 3 columns):
# #   Column  Non-Null Count  Dtype
#---  ------  --------------  -----
# 0   area    3 non-null      object
# 1   tel     3 non-null      int64    # 電話番号のはずが整数として扱われている
# 2   value   3 non-null      float64
```

ご覧の通り、電話番号としていたはずが整数値として扱われている（先頭の「0」がなくなっていることからも分かるはずだ）。これを回避するには、幾つかの方法がある。例えば、全ての値を文字列としてしまうことが考えられる。

```
df = pd.read_csv(filepath, dtype=str)  # 全てのデータの型を str に
print(df)
#       area         tel value
#0     tokyo  0312345678   1.0
#1  kanagawa   045678901   2.0
#2     chiba   043210987   3.0
```

「dtype=str」とすることで、全てのデータを文字列化したので、先頭の「0」が削除されなくなった。とはいえ、列ごとにデータ型を指定できた方がよいだろう。その場合は、以下のように、各列のデータ型を指定できる。

```
df = pd.read_csv(filepath, dtype={0: str, 1: str, 2: float})
print(df)  # 出力は省略
```

ここでは、全ての列のデータ型を指定しているが、特定の列だけを指定してもよい。

日付のパース

CSV ファイルに日付データが含まれている場合には、parse_dates パラメーターを指定することで、日付型のデータへとパースできる。

例えば、以下のようなデータがあったとする。

```
filepath = 'test5.csv'
print(Path(filepath).read_text())
#date,value0,value1
#2021/09/07,1.0,2.0
#2021/09/08,3.0,4.0
#2021/09/09,5.0,5.0
```

parse_dates パラメーターにはブール値、整数リスト、列名を要素とするリスト、辞書を渡せる。最初にこのパラメーターに True を渡す例を示す。

```
df = pd.read_csv(filepath, parse_dates=True, index_col=0)
print(df)
#            value0  value1
#date
#2021-09-07     1.0     2.0
#2021-09-08     3.0     4.0
#2021-09-09     5.0     5.0
```

「parse_dates=True」を指定すると、行インデックスとなる列の値が日付型に変換される（そのため、ここでは「index_col=0」を指定している）。

あるいは、変換したい列を表す整数値、または文字列（列名）を要素とするリストを渡してもよい。以下に例を示す（変換したい列が 1 つだけでもリストで渡す必要がある）。

```
df = pd.read_csv(filepath, parse_dates=[0])
print(df)
#        date  value0  value1
#0 2021-09-07     1.0     2.0
#1 2021-09-08     3.0     4.0
#2 2021-09-09     5.0     5.0

df = pd.read_csv(filepath, parse_dates=['date'])
print(df)  # 出力は省略
```

1 つ目の例では「parse_dates=[0]」として、2 つ目の例では「parse_dates=['date']」として日付を含んだ列を指定している。

日付を表すデータが複数の列に分かれていることもあるかもしれない。その場合は、parse_dates パラメーターに一つの日付を表す列をリストにまとめ、それをさらにリストの要素として渡すとよい。

例えば、以下のように year 列、month 列、date 列に分けて、日付のデータが格納されていたとする。

```
filepath = 'test6.csv'
print(Path(filepath).read_text())
#year,month,day,value0,value1
#2021,9,7,1.0,2.0
#2021,9,8,3.0,5.0
#2021,9,9,4.0,6.0
```

このときに、それらをまとめて日付型に変換するには以下のようにする。

```
dates = [['year', 'month', 'day']]
df = pd.read_csv(filepath, parse_dates=dates)
print(df)
#  year_month_day  value0  value1
#0     2021-09-07     1.0     2.0
#1     2021-09-08     3.0     5.0
#2     2021-09-09     4.0     6.0
```

列名が「year_month_day」と 3 つの列をつなげたものになっている点に注意しよう。辞書を渡すと、次のように列名を指定できる。

```
dates = {'date': ['year', 'month', 'day']}
df = pd.read_csv(filepath, parse_dates=dates)
print(df)
#        date  value0  value1
#0 2021-09-07     1.0     2.0
#1 2021-09-08     3.0     5.0
#2 2021-09-09     4.0     6.0
```

この例では日付は 1 つだけなので、辞書を使う意味が分からないかもしれないが、開始日と終了日など複数の日付が含まれている場合に、「dates = {'start_date': [……], 'end_date': [……]}」などと書けることは覚えておこう。

欠損値の扱い

CSV ファイルには空文字列や「nan」「NA」などとして値がないことを示すデータが含まれているかもしれない。read_csv 関数はデフォルトで、そうした値を NaN として扱ってくれる（NaN 値については「CSV ファイルから読み込みを行うには（NumPy 編）」のコラムを参照）。

例えば、次のような CSV ファイルがあったとする。

```
filepath = 'test7.csv'
print(Path(filepath).read_text())
# col0,col1,col2
# ,nan,1.0
# 2.0,N/A,null
# NaN,3.0,--
```

CSV ファイルには「nan」「N/A」などが欠損値があるフィールドとして記述されている。これを read_csv 関数で読み込むと次のようになる。

```
df = pd.read_csv(filepath)
print(df)
#   col0  col1 col2
#0   NaN   NaN  1.0
#1   2.0   NaN  NaN
#2   NaN   3.0   --  # 「--」という文字列は欠損値としては扱われていない
```

多くの値は NaN に変換されたが、CSV ファイルにあった「--」というフィールドはそうはなっていない点に注意。これも欠損値として扱いたければ、na_values パラメーターにそれを指定すればよい。

```
df = pd.read_csv(filepath, na_values=['--'])
print(df)
#   col0  col1  col2  # デフォルトの欠損値と na_values に指定した値が欠損値
#0   NaN   NaN   1.0
#1   2.0   NaN   NaN
#2   NaN   3.0   NaN
```

pandas がデフォルトで欠損値として扱っている文字列を無効化するには、keep_default_na パラメーターに False を指定する（以下の例では左下にも「NaN」があるが、これは文字列である）。

```
df = pd.read_csv(filepath, keep_default_na=False,  na_values=['--'])
print(df)
#  col0 col1  col2  # na_values に指定した値のみが欠損値として扱われる
#0       nan   1.0
#1  2.0  N/A  null
#2  NaN  3.0   NaN
```

［解決！Python］CSV ファイルに書き込みを行うには（pandas 編）

pandas.DataFrame クラスの to_csv メソッドを使って、データフレームの内容を CSV ファイルに書き込む方法を紹介する。

（2021 年 09 月 14 日）

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

data = {
    'name': ['isshiki', 'endo', 'kawasaki'],
    'age': [20, 25, np.nan],
    'weight': [55.44, 66.77, 123.456]
}
df = pd.DataFrame(data)
print(df)
#       name   age   weight
#0   isshiki  20.0   55.440
#1      endo  25.0   66.770
#2  kawasaki   NaN  123.456

# 基本
fname = 'test.csv'
df.to_csv(fname)
print(Path(fname).read_text())
#,name,age,weight
#0,isshiki,20.0,55.44
#1,endo,25.0,66.77
#2,kawasaki,,123.456

# 区切り文字の変更
df.to_csv(fname, sep=' ')
print(Path(fname).read_text())
# name age weight
```

```
#0 isshiki 20.0 55.44
#1 endo 25.0 66.77
#2 kawasaki  123.456

# 欠損値の表現を指定する
df.to_csv(fname, na_rep='nan')
print(Path(fname).read_text())
#,name,age,weight
#0,isshiki,20.0,55.44
#1,endo,25.0,66.77
#2,kawasaki,nan,123.456

# 数値を文字列化する際の書式指定
df.to_csv(fname, float_format='%+08.3f')
print(Path(fname).read_text())
#,name,age,weight
#0,isshiki,+020.000,+055.440
#1,endo,+025.000,+066.770
#2,kawasaki,,+123.456

# ヘッダー行の指定
df.to_csv(fname, header=['col0', 'col1', 'col2'])
print(Path(fname).read_text())
#,col0,col1,col2
#0,isshiki,20.0,55.44
#1,endo,25.0,66.77
#2,kawasaki,,123.456

# 行インデックスの列名を指定
df.to_csv(fname, index_label='idx')
print(Path(fname).read_text())
#idx,name,age,weight
#0,isshiki,20.0,55.44
#1,endo,25.0,66.77
#2,kawasaki,,123.456
```

```
# 行インデックスを出力しない
df.to_csv(fname, index=False)
print(Path(fname).read_text())
#name,age,weight
#isshiki,20.0,55.44
#endo,25.0,66.77
#kawasaki,,123.456

# 書き出す列の指定
df.to_csv(fname, columns=['name', 'weight'])
print(Path(fname).read_text())
#,name,weight
#0,isshiki,55.44
#1,endo,66.77
#2,kawasaki,123.456

# クオートの指定
import csv
df.to_csv(fname, quoting=csv.QUOTE_ALL)
print(Path(fname).read_text())
#"","name","age","weight"
#"0","isshiki","20.0","55.44"
#"1","endo","25.0","66.77"
#"2","kawasaki","","123.456"

df.to_csv(fname, quoting=csv.QUOTE_NONNUMERIC, quotechar="'")
print(Path(fname).read_text())
#'','name','age','weight'
#0,'isshiki',20.0,55.44
#1,'endo',25.0,66.77
#2,'kawasaki','',123.456

# フィールドを囲むのに使う引用符自体がフィールドに含まれている場合の処理
data = {
    "name": ["chak'n", "and pop"],
```

```
    "value": [100, 120]
}
df = pd.DataFrame(data)
print(df)
#       name  value
#0   chak'n    100
#1  and pop    120

df.to_csv(fname, quoting=csv.QUOTE_NONNUMERIC, quotechar="")
print(Path(fname).read_text())
#","name","value"
#0,"chak""n",100
#1,"and pop",120

df.to_csv(fname, sep=' ', quoting=csv.QUOTE_NONE, escapechar='¥')
print(Path(fname).read_text())
# name value
#0 chak'n 100
#1 and¥ pop 120
```

pandas.DataFrame.to_csv メソッド

pandas が提供する DataFrame クラスには to_csv メソッドがあり、これを使うことでデータフレームに格納されているデータを CSV ファイルへと書き出せる。

以下は本稿で紹介するパラメーターを含む、to_csv メソッドの基本的な構文だ。

```
pandas.DataFrame.to_csv(path, sep, na_rep, float_format, columns, header,
                        index, index_label, quoting, quotechar)
```

本稿では以下のパラメーターを紹介する。全てのパラメーターについては、pandas のドキュメント「pandas.DataFrame.to_csv」を参照のこと。

- path：データフレームの内容を書き出すファイルの名前。必須
- sep：区切り文字。省略可。省略時はカンマ「,」が指定されたものとして扱われる
- na_rep：欠損値を表す文字列表現。省略可。省略時は空文字列「“”」が指定したものとして扱われる
- float_format：数値を文字列化してファイルに書き込む際に使われる書式化指定文字列。省略可。省略時は元の値がそのまま文字列化される
- columns：どの列の値を書き込むかの指定。省略可。省略時は行インデックスに続いて、全ての列が書き込まれる。
- header：ヘッダー行の指定。True ／ False、文字列リストを指定可能。文字列リストを指定した場合は、それらがデータフレームの列名の代わりにヘッダー行に使われる。False を指定した場合は、ヘッダー行は書き込まれない。True を指定した場合は、データフレームの列名がヘッダー行に書き込まれる。省略可。省略時は True が指定されたものとして扱われる
- index：行インデックスを書き込むかどうかを指定する（True ／ False）。省略可。省略時は True が指定されたものとして扱われる（行インデックスが書き込まれる）
- index_label：行インデックスの列名としてヘッダー行に書き込む値を指定する。header パラメーターと index パラメーターが True の場合に、index_label パラメーターに渡した値が行インデックスの列名としてヘッダー行に書き込まれる。省略可。省略時は行インデックスの列名は書き込まれない
- quoting：CSV ファイルに書き込む各フィールドを何らかの引用符で囲むかどうかを指定する。指定可能なのは、Python に標準で添付される csv モジュールで定義されている定数
- quotechar：各フィールドを囲む引用符の指定。長さ 1 の文字。省略可。省略時はダブルクオート「“」が指定されたものとして扱われる

基本的な使い方

以下では、次のコードで作成したデータフレームを CSV ファイルに書き込むものとする。

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

data = {
    'name': ['isshiki', 'endo', 'kawasaki'],
    'age': [20, 25, np.nan],
    'weight': [55.44, 66.77, 123.456]
}
df = pd.DataFrame(data)
print(df)
#       name   age   weight
#0   isshiki  20.0   55.440
#1      endo  25.0   66.770
#2  kawasaki   NaN  123.456
```

一番簡単なのは、書き込むファイル名を指定して、このメソッドを呼び出すだけだ。以下に例を示す。

```
fname = 'test.csv'
df.to_csv(fname)
print(Path(fname).read_text())
#,name,age,weight
#0,isshiki,20.0,55.44
#1,endo,25.0,66.77
#2,kawasaki,,123.456
```

出力結果を見ると、ヘッダー行の先頭には行インデックスの列名がない、各行の先頭に行インデックスがある、区切り文字がカンマ「,」となっている、欠損値があれば空文字列が書き込まれることなどが分かるはずだ。

区切り文字の変更

区切り文字を変更するには sep パラメーターに長さ 1 の文字を指定する。以下に例を示す。

```
df.to_csv(fname, sep=' ')
print(Path(fname).read_text())
# name age weight
#0 isshiki 20.0 55.44
#1 endo 25.0 66.77
#2 kawasaki  123.456
```

上の例では「sep=' '」としているので半角空白文字で各フィールドが区切られている。

欠損値の表現を指定する

欠損値を含むフィールドを CSV ファイルに書き込む際に、どんな文字列表現とするかを指定するには na_rep パラメーターにその値を指定する。以下に例を示す。

```
df.to_csv(fname, na_rep='nan')
print(Path(fname).read_text())
#,name,age,weight
#0,isshiki,20.0,55.44
#1,endo,25.0,66.77
#2,kawasaki,nan,123.456
```

この例では、「na_rep='nan'」としているので、CSV ファイルの該当するフィールドには「nan」が書き込まれている。

数値を文字列化する際の書式指定

数値を文字列化して CSV ファイルに書き込む際に、どのような形式に書式化するかは float_format パラメーターで指定できる。以下に例を示す。

```
df.to_csv(fname, float_format='%+08.3f')
print(Path(fname).read_text())
#,name,age,weight
#0,isshiki,+020.000,+055.440
#1,endo,+025.000,+066.770
#2,kawasaki,,+123.456
```

書式化指定文字列は「%」で始まり、その後に（符号の付加、左寄せなどを指定する）フラグとフィールドの最小文字数、さらに「.」の後に小数点以下の精度と（整数、小数、指数表記などを示す）指定子が続く。

上の例では「+」は常に符号を付加するフラグで、「0」は 0 埋めを表すフラグ、「8」がフィールドの最小文字数、「3」は小数点以下 3 桁までを表示する指示、最後の「f」が小数点表示を意味している。CSV ファイルの内容と付き合わせてみてほしい。

ヘッダー行の指定

データフレームの列名と CSV ファイルのヘッダー行とを異なるものにしたければ、header パラメーターにヘッダー行の内容を指定する。以下に例を示す。

```
df.to_csv(fname, header=['col0', 'col1', 'col2'])
print(Path(fname).read_text())
#,col0,col1,col2
#0,isshiki,20.0,55.44
#1,endo,25.0,66.77
#2,kawasaki,,123.456
```

なお、ヘッダー行を出力したくないときには、「header=False」とすればよい（例は省略）。

行インデックスの列名を指定

行インデックスの列名をヘッダー行に含めたいときには、index_label パラメーターを使用する。以下はその例だ。

```
df.to_csv(fname, index_label='idx')
print(Path(fname).read_text())
#idx,name,age,weight
#0,isshiki,20.0,55.44
#1,endo,25.0,66.77
#2,kawasaki,,123.456
```

なお、このパラメーターは header パラメーターが False 以外のとき、かつ以下で紹介する index パラメーターが True のときにだけ機能する（つまり、ヘッダー行を出力し、行インデックスを出力するときだけ、このパラメーターに指定した列名も出力される）。

行インデックスを出力しない

これまでに見てきた通り、to_csv メソッドはデフォルトで行インデックスを（行の先頭に）出力する。これをやめたいときには、index パラメーターに False を指定する。以下に例を示す。

```
df.to_csv(fname, index=False)
print(Path(fname).read_text())
#name,age,weight
#isshiki,20.0,55.44
#endo,25.0,66.77
#kawasaki,,123.456
```

書き出す列の指定

特定の列だけを CSV ファイルに書き込みたければ、その列名を要素とするリスト（シーケンス）を columns パラメーターに指定する。以下に例を示す。

```
df.to_csv(fname, columns=['name', 'weight'])
print(Path(fname).read_text())
#,name,weight
#0,isshiki,55.44
#1,endo,66.77
#2,kawasaki,123.456
```

クオートの指定

to_csv メソッドでは基本的には各フィールドを何らかの引用符でなるべく囲まないようにしている（Python に標準添付される csv モジュールの csv.QUOTE_MINIMAL 値に対応）。これを変更するには、quoting パラメーターに csv モジュールで定義されている値を指定する。

以下の例では、全てのフィールドを囲むように指定している。

```
import csv
df.to_csv(fname, quoting=csv.QUOTE_ALL)
print(Path(fname).read_text())
#"","name","age","weight"
#"0","isshiki","20.0","55.44"
#"1","endo","25.0","66.77"
#"2","kawasaki","","123.456"
```

デフォルトでは引用符にダブルクオート「"」が使われるので、上のような出力結果となる。引用符を変更するには、quotechar パラメーターに長さ 1 の文字を指定する。

以下は数値以外のフィールドをシングルクオート「'」で囲むように指示する例だ。

```
df.to_csv(fname, quoting=csv.QUOTE_NONNUMERIC, quotechar="'")
print(Path(fname).read_text())
#'','name','age','weight'
#0,'isshiki',20.0,55.44
#1,'endo',25.0,66.77
#2,'kawasaki','',123.456
```

なお、フィールドを囲むのに使う引用符自体がフィールドに含まれていた場合には、それらは自動的に二重化される（その文字を二度繰り返す）。

例えば、以下のようなデータフレームがあったとする。

```
data = {
    "name": ["chak'n", "and pop"],
    "value": [100, 120]
}
df = pd.DataFrame(data)
print(df)
#       name  value
#0    chak'n    100
#1  and pop     120
```

これを CSV ファイルに（数値以外のフィールドを）シングルクオートで囲んで出力するとしたらどうなるだろう。「chak'n」というフィールドに注目されたい。

```
df.to_csv(fname, quoting=csv.QUOTE_NONNUMERIC, quotechar="'")
print(Path(fname).read_text())
#'','name','value'
#0,'chak''n',100
#1,'and pop',120
```

出力を見ると「chak''n」とシングルクオートが二重化されていることが分かる。

また、以下は区切り文字を空白文字として、CSV ファイルに書き込みを行おうとするコードだが、今度は「and pop」というフィールドに空白文字が含まれていることが分かる。二重化するのではなく、何らかの文字でエスケープしたいときには次のように escapechar パラメーターが使える。

```
df.to_csv(fname, sep=' ', quoting=csv.QUOTE_NONE, escapechar='¥¥')
print(Path(fname).read_text())
# name value
#0 chak'n 100
#1 and¥ pop 120
```

escapechar パラメーターは、doublequote パラメーターと組み合わせて使うこともできる。doublequote パラメーターは今見た二重化を制御するパラメーターでデフォルト値は True（二重化する）になっている。これを False にして escapechar パラメーターと組み合わせた例も示しておこう。

```
df.to_csv(fname, quoting=csv.QUOTE_ALL, quotechar="'", doublequote=False,
escapechar='¥¥')
print(Path(fname).read_text())
#"','name','value'
#'0','chak¥'n','100'
#'1','and pop','120'
```

編集：@IT 編集部
発行：アイティメディア株式会社