SECTION 005

どちらを使う? VBAとPythonの使い分けのポイント

ここまで説明してきた通り、Excelのデータを自動的に処理するには、VBAとPythonのどちらも利用できます。ここで、改めてこの両者を比較し、目的や状況に応じて、それぞれどちらを使用すればよいかを考察していきましょう。

▫「Excelの自動化」の2つの側面

「Excelを自動化する」と一口にいっても、その具体的な内容には、大きく分けて、次の2つの側面があります。1つは「Excelの操作の自動化」、もう1つは「Excelデータの処理の自動化」です。

Excelのアプリケーション自体は、いうまでもなく、ユーザーが作業をするための環境です。Excelで自分自身が作業したり、他の人に指示をして作業させたりする際、その省力化や正確性の向上を図るために、一連の手順を自動化するというアプローチが、「Excelの操作の自動化」です。

一方で、Excelのデータは、ビジネスのさまざまな分野で広く利用されている、標準的な「データフォーマット」でもあります。重要なのはExcel形式のデータであり、それを処理するための方法は、必ずしもExcelでなければならないわけではありません。特に、過去に蓄積された大量のExcelデータに対しては、その1つ1つをユーザーがExcelで開いて作業するよりも、何らかのプログラムで、各ファイルを連続で開いて自動処理したほうが効率的です。これが、「Excelデータの処理の自動化」ということです。

▫VBAを利用するとよいケース

目的が「Excelの操作の自動化」であれば、Excel自体の備える機能のほとんどをコード化でき、またそのプログラムをExcel自体の機能であるかのように追加できるVBAのほうが、自動化の仕組みとしては優れています。

Excelでの作業がメインですが、重要な箇所では人間の判断や意思決定が必要になる場合、その作業全体を自動化することはできません。作業の一部分だけを効率化するために自動化を図りたいのであれば、PythonよりもVBAを利用することをおすすめします。前述したように、VBAでは、いわばExcelに独自のコマンドを追加するかのように機能を追加できるからです。

逆に、Excelでの作業と並行して、同じブックをPythonで処理しようとするのは、何かと面倒なことも少なくありません。たとえば、openpyxlでの処理は、対象のブックをExcelで開いている状態では実行できないため、一度閉じて実行してから、改めて開き直す必要があります。

また、Excelの数式で使えるユーザー定義関数を追加したり、Excelの操作に応じて自動的にプログラムを実行したりといったことも、VBAを使用するメリットの1つです。Pythonでユーザー定義関数を作る方法もないわけではありませんが、いずれにしてもVBAの知識がある程度は必要です。

さらに、自分自身が作業する場合以外にも、Excelでフォーマットを作り、そのブックで他の人に作業してもらうというケースもよくあります。このようなブックでの作業をより便利にする仕組みを作る際も、やはりVBAを利用するのがよいでしょう。たとえば、リボンやクイックアクセスツールバーにボタンを追加し、それをクリックするだけで、選択範囲の各セルを対象としたVBAのマクロを実行する、といったことが可能です。

▫Pythonを利用するとよいケース

必ずしもExcelで実際に作業する必要はなく、目的が「Excelデータの処理の自動化」である場合は、VBAにこだわる必要はありません。蓄積された大量のExcelデータを一気に処理したい場合は、Pythonのプログラムを利用することで、逆にExcelを一切使用することなく処理を完了できます。

また、Pythonを利用することで、Excel以外の処理との連携がよりスムーズになる可能性があります。作業の各部分で人間の判断や意思決定が必要なく、Excelやその他のプログラムの機械的な処理を連続で実行したい場合は、Pythonを利用して一連の処理を完全に自動化することも可能でしょう。

前述した通り、Pythonには、さまざまな処理を容易に実行するための豊富なライブラリが用意されています。Excelデータの処理とそれらのライブラリを組み合わせることで、Excel単体では実現不可能な、より多彩で複雑な処理が可能になります。

なお、Excel VBAで大量データの処理ができないわけではありませんが、処理対象のブックそのものにプログラムを記述するのは、やはりいろいろとデメリットがあります。プログラム専用のブックを作成し、処理対象のブックはその外部に置いて、プログラムから開く形で処理したほうがいいでしょう。

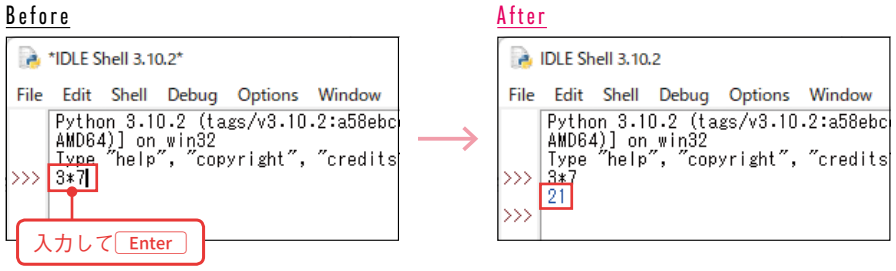
SECTION 009

かんたんな計算をしてみよう

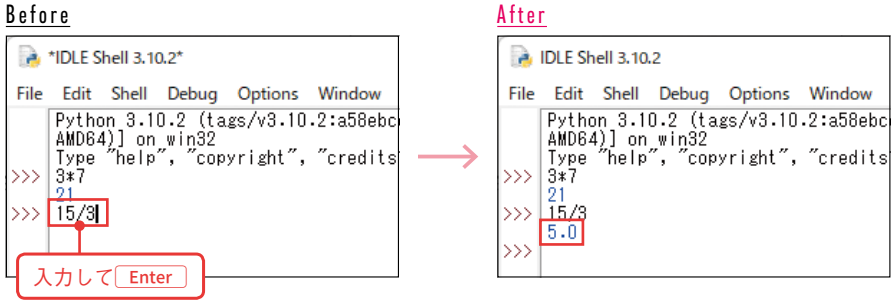
前述したように、IDLEのShell画面では、式を入力して、その計算結果を求めることができます。ここではShell画面上でさまざまな計算をする過程を通して、Pythonでどのような計算処理が可能なのかを見ていきましょう。

□ 数値データを演算する

まず、数値同士の基本的な四則計算を実行します。IDLEのShellで、「>>>」の後に「3 * 7」と入力して、Enterキーを押します。これで、3と7の積が求められます。



次に、「>>>」の後に「15 / 3」と入力してEnterキーを押してみましょう。15を3で割った商が求められます。



割り切れる計算なのに「5」ではなく「5.0」となっているのは、除算の結果は自動的にfloat型（浮動小数点数型）の数値になるためです。

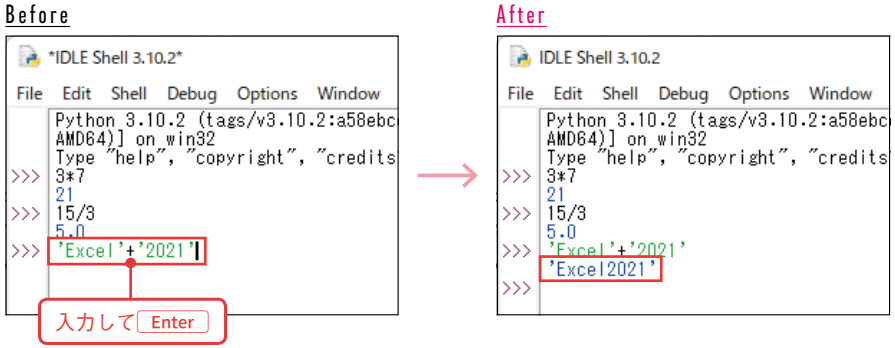
乗算の「*」や除算の「/」などを「算術演算子」と呼びます。Pythonで使用できる主な算術演算子には次のような種類があります。

演算子	使用法	演算の内容
+	数値a + 数値b	数値aに数値bを加えた和を求める
-	数値a - 数値b	数値aから数値bを引いた差を求める
*	数値a * 数値b	数値aに数値bを掛けた積を求める
/	数値a / 数値b	数値aを数値bで割った商を求める
%	数値a % 数値b	数値aを数値bで割った余りを求める
//	数値a // 数値b	数値aを数値bで割った整数部分を求める
**	数値a ** 数値b	数値aの数値b乗を求める

□ 文字列データを処理する

Pythonでは文字列データを処理することもできます。P.38でも説明した通り、文字列は「'」または「"」で囲んで指定し、並べて指定することで結合できます。ただし、この方法が使えるのは、文字列そのもののデータの場合です。変数（P.46参照）に収めた文字列を、この方法で結合することはできません。

「+」は、数値データではなく文字列データを対象とした場合には、結合の文字列演算子になります。この方法では、文字列そのものでも変数に収めた文字列でも、問題なく結合できます。たとえば、「>>>」の後に「'Excel' + '2021'」と入力してEnterキーを押すと、「'Excel2021'」と表示されます。



なお、この例では「2021」も「'」で囲んで文字列として指定しています。「'」で囲まない場合は数値として扱われ、この処理ではエラーになるので注意が必要です。

SECTION 026

セルから数式と計算結果の値を取り出そう

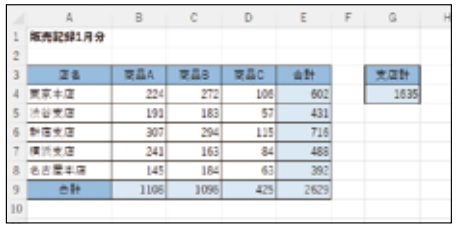
前項では、指定したセルのデータを取り出す方法を解説しましたが、対象のセルの内容が数式だった場合は、その数式がそのまま取り出されます。ここでは、セルの数式ではなく、その計算結果の値を取り出す方法を紹介します。

□ セルの数式を取り出す

前項で説明した通り、特定のセルを表すオブジェクトを取得したら、そのvalueで、そのセルの内容を取り出すことができます。ただし、対象のセルの内容が値ではなく数式だった場合、value（値）とはいっても、その数式が取り出されます。

今回も、実行中のスクリプトファイルと同じフォルダーにあるブック「販売記録01.xlsx」からデータを取り出します。

ブック「販売記録01.xlsx」



	A	B	C	D	E	F	G
1	販売記録1月分						
2							
3	店名	商品A	商品B	商品C	合計		売上計
4	東京本店	224	272	106	602		1635
5	渋谷支店	193	183	57	431		
6	新宿支店	307	294	115	716		
7	横浜支店	241	163	84	488		
8	名古屋本店	145	184	63	392		
9	合計	1108	1096	425	2629		
10							

このアクティブシートのセル範囲B9:D9には同じ列の上側5行分、セル範囲E4:E9には同じ行の左側3列分のセルの合計を求める数式が、それぞれ入力されています。

前項と同様のプログラムでこのセルE4のデータを取り出した場合、その数式がそのまま取り出されます。

PROGRAM ▸ sample026_1.py

```
import openpyxl
wb = openpyxl.load_workbook('販売記録01.xlsx')
ws = wb.active
print(ws['E4'].value)
```

実行例

```
==== RESTART: C:¥Users¥clayh¥Documents¥Works¥ExcelPython¥3章作例¥sample026_1.py
===
=SUM(B4:D4)
>>>
```

□ セルの計算結果の値を取り出す

指定したセルの数式ではなく、その計算結果の値を取り出すことも可能です。次の例は、アクティブシートのセルE4の数式の計算結果の値を出力するプログラムです。

PROGRAM ▸ sample026_2.py

```
import openpyxl
wb = openpyxl.load_workbook('販売記録01.xlsx', data_only=True)
ws = wb.active
print(ws['E4'].value)
```

実行例

```
==== RESTART: C:¥Users¥clayh¥Documents¥Works¥ExcelPython¥3章作例¥sample026_2.py
===
602
>>>
```

実は、openpyxlでは、1回読み込んだブックから、セルの数式とその計算結果の値の両方を取り出す、簡単な方法はありません。

計算結果の値を取り出したい場合は、load_workbookでブックを読み込むときに、引数として「data_only=True」を指定します。この方法で読み込んだ場合は、逆にセルのデータを数式として取り出すことはできなくなるので、数式と値の両方が必要な場合は、それぞれの目的のために、ブックを読み込み直す必要があります。

SECTION 042

住所を地名と数字部分に分けよう

住所を表す文字列が、1つのセルに入力されています。その地名の部分と、番地や号を表す数字の部分とに分割して、それぞれ同じ行の別のセルに入力するプログラムを紹介します。つまり、文字列中で最初に現れた数字の位置で分割する処理です。

□ 住所を地名と番地に分割する

今回は、営業所の情報が入力されたブック「営業所情報01.xlsx」を作業対象とします。そのアクティブシートの2列目（B列）に入力された住所の文字列を、市区町村名を表す「地名」部分と、番地・号の数字部分に分割して、それぞれ右側の2列に入力するプログラムを作成します。なお、住所の文字列の途中に現れた算用数字以降を番地・号と見なすので、番地を漢数字で入力している場合は正しく分割できません。

ブック「営業所情報01.xlsx」

PROGRAM ▸ sample042_1.py

```
import openpyxl
import re
fname = '営業所情報01.xlsx'
wb = openpyxl.load_workbook(fname)
ws = wb.active
for row in ws.iter_rows(min_row=4):
    s = row[1].value
    result = re.match(r'(.+?)(¥d.*)', s)
    if result:
        row[2].value = result.group(1)
        row[3].value = result.group(2)
wb.save(fname)
```

実行例

for文で、対象のワークシートの4行目以降の入力済みの行を、行単位で繰り返し処理します。その各行の2列目（B列）のセルの文字列を取り出し、変数sに代入します。

この変数sの文字列を対象にreのmatchを実行し、正規表現のパターンとして「(.+?)(¥d.*)」という文字列を指定します。それぞれの「( )」の中は、1文字以上の任意の文字列と、数字から始まる任意の文字列の2つのグループを表しています。つまり、先頭からの一連の数字以外の文字列と、途中に出てきた数字以降の一連の文字列を、それぞれグループとしてまとめているわけです。

変数sの文字列がこのパターンにマッチしたかどうかをif文で判定し、Trueの場合は、その結果を取得した変数resultの1番目のグループを3列目（C列）に、2番目のグループを4列目（D列）のセルに入力します。

COLUMN

reモジュールの活用

ここまで、reモジュールで文字列のパターンを判定する機能（メソッド）として、「match」と「search」を使用する例を紹介してきました。ここで改めて、これらの使い分けについて説明しておきましょう。これらのメソッドは、いずれもマッチした部分の情報がマッチオブジェクトとして返され、「group」によってその文字列を取り出せます。異なっているのは、matchが対象の文字列の先頭からマッチするかどうかをチェックするのに対し、searchは文字列の途中からでもマッチする部分があるかどうかをチェックするという点です。そのため、セルの文字列が特定の形式に当てはまるかを調べるには、searchよりもmatchの方が適しています。ただし、文字列そのものに完全にマッチしているわけではなく、あくまでも先頭からの判定なので、マッチした部分以降に別の文字列がある可能性はあります。文字列全体とマッチしているかを判定したい場合は、「fullmatch」を使用します。

また、これらのメソッドでは、文字列の中にパターンにマッチする部分が2カ所以上あった場合も、最初に見つかった1カ所だけを表すマッチオブジェクトが返されます。マッチする文字列をすべて取得したい場合は、「findall」を使用します。このメソッドの戻り値は、マッチオブジェクトではなく、マッチした文字列が取り出されたリストです。

SECTION 066

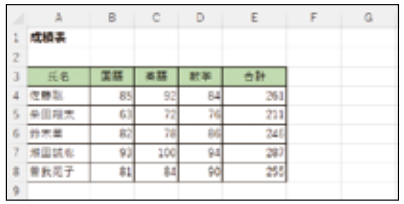
セルに条件付き書式を設定しよう

Excelの「条件付き書式」では、対象の範囲内で、条件を満たすセルの書式を自動的に変化させることができます。ここでは、Pythonのプログラムで、セルの値に応じて書式を変化させたり、セル内にデータバーを表示させたりする条件付き書式を設定します。

□ 指定値以上のルールを設定する

次のブック「成績表03.xlsx」のアクティブシートのセル範囲B4:D8を対象に、点数が90点以上だった場合はそのセルの数字を濃い青の太字で表示する条件付き書式を設定してみましょう。

ブック「成績表03.xlsx」

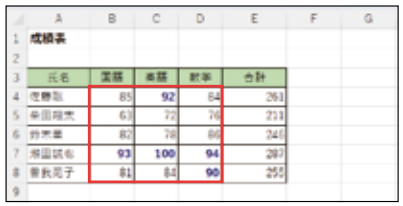


openpyxlで基本的な条件付き書式を設定するには、各設定方法を引数によって指定できるRuleクラスを使用する方法と、「セルの値」専用のCellIsRuleクラスを使用する方法とがあります。ここでは、比較的シンプルに記述できる、CellIsRuleクラスを使用する方法を紹介します。

PROGRAM ▸ sample066_1.py

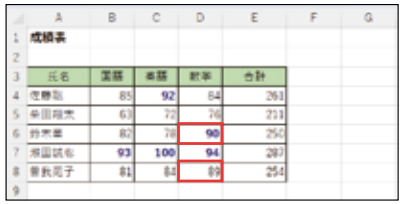
```
import openpyxl
from openpyxl.styles import Font
from openpyxl.formatting.rule import CellIsRule
fname = '成績表03.xlsx'
wb = openpyxl.load_workbook(fname)
ws = wb.active
fnt = Font(color='00008B', bold=True)
nrule = CellIsRule(operator='greaterThanOrEqual', formula=[90], ↩
                   stopIfTrue=None, font=fnt)
ws.conditional_formatting.add('B4:D8', nrule)
wb.save(fname)
```

実行例



条件付き書式は通常の書式とは異なり、セルの値を変更したり、参照しているセルの変更によって数式の結果が変わったりすると、自動的にその書式が変化します。

データを変更（参考例）



条件に応じて変化する書式として、ここではフォントに関する書式を指定します。そのため、まず「Font」をインポートします。さらに、条件付き書式に関する機能として「formatting.rule」に含まれる「CellIsRule」をインポートしています。

これまでと同様の手順で対象のブックを開き、そのアクティブシートを表すオブジェクトを変数wsに収めます。

Fontで変化後のフォントの書式を指定して、変数fntに代入します。そして、CellIsRuleで条件付き書式の設定を定義し、その引数「font」に指定することで、変化する書式として設定できます。また、引数「operator」で条件の設定方法を、引数「formula」でその設定内容を指定します。引数operatorでは、次のような指定が可能です。

指定値	指定内容	指定値	指定内容
between	次の値の間	greaterThan	次の値より大きい
notBetween	次の値の間以外	lessThan	次の値より小さい
equal	次の値に等しい	greaterThanOrEqual	次の値以上
notEqual	次の値に等しくない	lessThanOrEqual	次の値以下

SECTION 070

テーブルにデータを追加入力しよう

ここでは、対象のブックに作成済みのテーブルに、指定したデータを追加するPythonのプログラムを紹介しましょう。ただし、この操作をopenpyxlで行うのはやや面倒です。ここでは、pywin32を使ってテーブルにデータを追加する方法を紹介します。

□ テーブルにデータを追加する

ブック「メンバー情報11.xlsx」のアクティブシートの「メンバー」というテーブルに行を追加し、一連のデータを追加入力します。

ブック「メンバー情報11.xlsx」

PROGRAM ▸ sample070_1.py

```
import os
import win32com.client
pname = os.path.dirname(__file__)
fname = os.path.join(pname, 'メンバー情報11.xlsx')
xlApp = win32com.client.Dispatch('Excel.Application')
wb = xlApp.Workbooks.Open(fname)
xlApp.Visible = True
tbl = wb.ActiveSheet.ListObjects('メンバー')
nrow = tbl.ListRows.Add()
rval = ['BD1243', '山本雄二', 27, '経営管理部','企画第3班']
nrow.Range.Value = rval
wb.Close(SaveChanges=True)
xlApp.Quit()
```

実行例

作成済みのテーブルに行を追加してデータを入力する操作は、openpyxlよりもpywin32を使って、VBAと同様の方法で実行するのが効率的です。Excelのアプリケーションを起動し、実行中のスクリプトファイルと同じフォルダーにある「メンバー情報11.xlsx」を表すブックを開いて、そのアクティブシートの「メンバー」というテーブルを表す「ListObject」オブジェクトを取得し、変数tblに代入します。そのテーブルに行を追加し、戻り値として新行を表す「ListRow」オブジェクトを取得して、変数nrowに代入します。

この行に入力したいデータは、テーブルの1行分のセル数と合わせてリスト化し、変数rvalに代入します。テーブルの新行を表すオブジェクトの「Range」でそのセル範囲を表す「Range」オブジェクトを取得し、その「Value」にリストを代入して、1行分のセル範囲に一括でデータを入力します。その後、ブックを保存して閉じ、Excelを終了します。

なお、Excelの通常の操作では、テーブルの最下行の下の行のセルにデータを入力するだけで、その行までテーブルが自動拡張されます。そのため、現在のテーブルのサイズがわかっている場合は、その行のセル範囲に直接入力するだけで、テーブルを拡張してデータを追加できます。たとえば、上のプログラムで追加されるセル範囲がA11:E11だとわかっている場合は、次のようなコードでもテーブルに追加できます。

PROGRAM ▸ sample070_2.py(一部)

```
rval = ['BD1243', '山本雄二', 27, '経営管理部','企画第3班']
wb.ActiveSheet.Range('A11:E11').Value = rval
```

同様の処理をopenpyxlで実行した場合、Excelを起動していないため、テーブルの自動拡張機能は働かず、単にテーブルの下にデータが入力されるだけです。

SECTION 096

すべてのワークシートのデータを処理しよう

ここからは、繰り返し処理（P.64参照）を利用して、複数のシートやブックに対し、同じ処理をまとめて実行する方法を紹介していきます。まず、指定したブックに含まれるすべてのシートの特定のセルの値を変更する操作を紹介しましょう。

□ 全シートのセルA1にシート名を入力する

まず、指定したブックに含まれるすべてのワークシートのセルA1の値を、その各シートのシート名と同じになるように変更してみましょう。

PROGRAM ▸ sample096_1.py

```
import openpyxl
fname = '店舗別12.xlsx'
wb = openpyxl.load_workbook(fname)
for ws in wb.worksheets:
    ws['A1'].value = ws.title
wb.save(fname)
```

実行例

対象のブックを開いて変数wbに収め、その「worksheets」でそのブックに含まれている各ワークシートを対象とした繰り返しを実行します。変数wsに収められた各シートのセルA1の「value」に、「title」で求めたそのシート名を代入することで、シート名をそのままセルA1に入力しています。

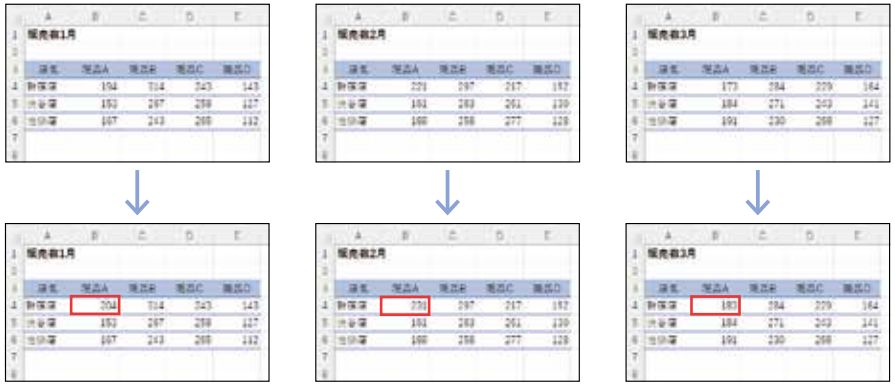
□ 各セルの同じセルの値に加算する

次に、対象のブックに含まれるすべてのワークシートのセルB4に入力された数値を、それぞれ現在の値に10を加えた値に変更してみましょう。

PROGRAM ▸ sample096_2.py

```
import openpyxl
fname = '店舗別12.xlsx'
wb = openpyxl.load_workbook(fname)
for ws in wb.worksheets:
    ws['B4'].value += 10
wb.save(fname)
```

実行例



複合代入演算子の「+=」などは、セルの値に代入する操作に対しても使用できます。これを利用して、現在のセルB4の値に10を加えた値を、改めて同じセルB4に代入しています。

SECTION 103

テキストファイルをシートに入力しよう

テキストファイルを読み込んで、Excelブックに入力するプログラムを作成します。ブックを新規作成して読み込むことも可能ですが、ここでは作成済みのブックのアクティブシートに番号付きで追加しましょう。

ブックにテキストを追加する

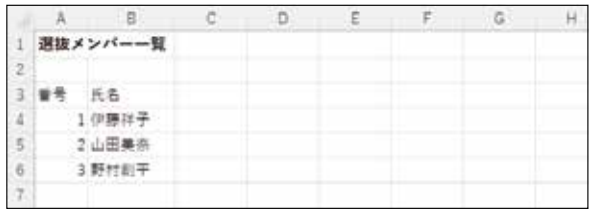
ここでは、「WorkData103」フォルダーの中にあるテキストファイル「追加メンバー.txt」を読み込み、同じフォルダーにあるブック「選抜メンバー01.xlsx」のアクティブシートの入力済みの行の末尾に、左側に番号を付けて追加します。番号は、現在の最下行の値を調べて、その続きからの連番にします。

「WorkData103」フォルダーの内容

追加メンバー.txt

```
斉藤雄太
吉岡弘道
水谷啓介
中村翠
竹内優香
朝倉亮二
```

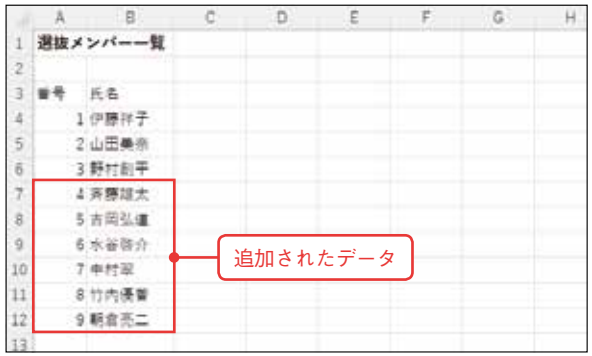
選抜メンバー01.xlsx



PROGRAM ▸ sample103_1.py

```
import openpyxl
tname = '追加メンバー.txt'
bname = '選抜メンバー01.xlsx'
wb = openpyxl.load_workbook(bname)
ws = wb.active
mnum = ws.cell(ws.max_row, 1).value
with open(tname, mode='r') as f:
    for line in f:
        mnum += 1
        nline = [mnum, line.rstrip('¥n')]
        ws.append(nline)
wb.save(bname)
```

実行例（ブック「選抜メンバー01.xlsx」）



「追加メンバー.txt」というテキストファイル名を変数tnameに、「選抜メンバー01.xlsx」というブック名を変数bnameに代入します。次に、変数bnameで指定したブックを開いて変数wbに収め、そのアクティブシートを変数wsに収めます。その「max_row」で入力済みの行の最大値を求め、その行の先頭（左端）のセルの値、つまり最下行のメンバーの番号を取り出します。

テキストファイルを開くには「open」を使用します。openの使い方はいくつかありますが、ここでは「with」の後にopenを指定し、その第1引数に変数tnameを、引数modeに読み込み用を意味する「r」を指定します。さらに「as f」と続けることで、読み込んだファイルを表すオブジェクトが変数fに代入されます。

for文の対象にこのファイルを表す変数fを指定することで、そのテキストを1行ずつ読み込んで変数lineに収め、以降の処理を繰り返します。各繰り返しでは、まず変数mnumに1を加算します。また、変数lineに収めた各行の文字列は、rstrip関数の引数に改行を表す「¥n」を指定することで、各行の末尾に付いている改行コードを取り除きます。変数mnumの値とこの文字列を、リストとして変数nlineに代入します。そして、この変数nlineを、appendで対象のワークシートの末尾に追加します。

最後に、このブックを上書き保存します。

SECTION 108

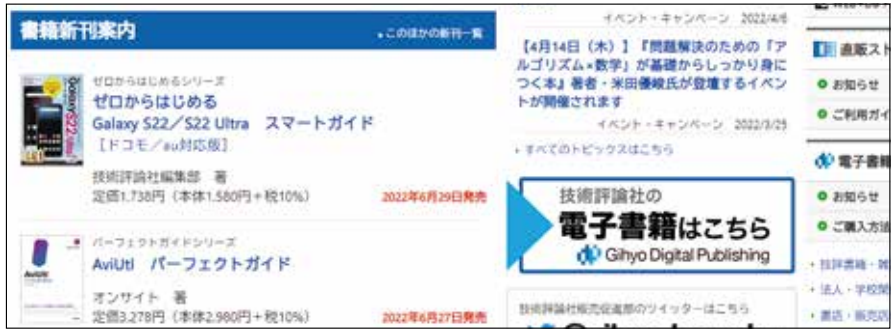
Webのデータをシートに入力しよう

Webページからデータを取得し、Excelのワークシートに自動入力してみましょう。対象のWebページの設計図であるHTMLの構成を解析し、目的のデータを取り出して、指定したセルに入力するプログラムを作成します。

▫ 指定URLから特定のデータを取り出す

ここでは、技術評論社の書籍紹介ページ（https://gihyo.jp/book）から、「書籍新刊案内」として表示されているすべての書名を取り出し、新規作成したブックに自動入力して、「新刊書籍一覧.xlsx」というファイル名で保存します。

書籍新刊案内のWebページ



あらかじめこのページのソースを調べて、そのHTMLがどのような構成で、取得したいデータがその中のどの要素かを確認しておきましょう。ここで取り出したい新刊書籍のタイトルは、id属性が「newBookList」であるdiv要素の中の、複数のh3要素です。

PythonでWebからデータを取得するには、標準ライブラリの「request」を使う方法もありますが、ここではより簡単に記述できる外部ライブラリの「requests」を使用します。また、取得したHTMLデータを解析して必要なデータを取り出す処理には、やはり外部ライブラリの「Beautiful Soup」を使用します。これらに加えて「lxml」を、事前にインストールしておく必要があります（P.83参照）。

コマンド

```
py -m pip install requests
```


コマンド

```
py -m pip install lxml
```

コマンド

```
py -m pip install beautifulsoup4
```

PROGRAM ▸ sample108_1.py

```
import openpyxl
import requests
from bs4 import BeautifulSoup
wb = openpyxl.Workbook()
ws = wb.active
ws['A1'].value = '新刊書籍リスト'
ws['A3'].value = '番号'
ws['B3'].value = '書名'
res = requests.get('https://gihyo.jp/book')
soup = BeautifulSoup(res.content, 'lxml')
newbooks = soup.find('div', id='newBookList')
for i, newbook in enumerate(newbooks.find_all('h3')):
    title = newbook.text
    line = [i + 1, title.replace('¥n', '')]
    ws.append(line)
wb.save('新刊書籍一覧.xlsx')
```

実行例（ブック「新刊書籍一覧.xlsx」）

	A	B	C	D	E	F	G	H	I	J	K	L
1	新刊書籍リスト											
2												
3	番号	書名										
4		1 ゼロからはじめるシリーズ ゼロからはじめるGalaxy S22／S22 Ultra スマートガイド［ドコモ／au対応版］										
5		2 パーフェクトガイドシリーズ AviUtl パーフェクトガイド										
6		3 情報処理技術者試験シリーズ 令和04年【下期】基本情報技術者 パーフェクトラーニング過去問題集										
7		4 はじめてでもできる Fusion 360入門										
8		5 図解即戦力シリーズ 図解即戦力UMLのしくみと実装がこれ1冊でしっかりわかる教科書										
9		6 大きな字でわかりやすいシリーズ 大きな字でわかりやすいGoogle グーグル入門										
10		7 レシピ集シリーズ Photoshopレタッチレシピ集										

まず、openpyxlの他、インストールした各モジュールをインポートします。そして、openpyxlの「Workbook」クラスで新規ブックを作成し、そのアクティブシートのセルA1にこのシートのタイトル「新刊書籍リスト」を、セルA3とセルB3にこの列のデータの見出しとして「番号」と「書名」と入力します。

requestsの「get」に、引数としてデータを取得したいWebページのURLを指定することで、リクエストに対するレスポンスボディを表すオブジェクトを取得できます。こ

SECTION 116

ファイルダイアログを表示しよう

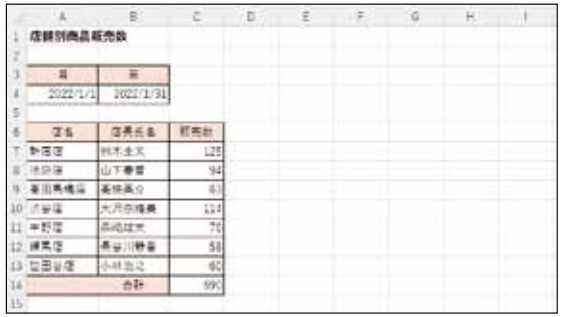
ここまでの例では、作業対象のExcelブックやテキストファイルなどは、プログラムの中で直接指定していました。ここでは、ファイルを選択するダイアログボックスを表示し、そのつどユーザーに作業対象のブックを選択させる例を紹介します。

□ 処理対象のファイルをダイアログで指定する

次のプログラムでは、「開く」ダイアログボックスを表示して、ユーザーがExcelブックまたはExcelマクロ有効ブックを選択すると、そのアクティブシート名と、シート内の数値セルの個数、およびそのすべての数値の合計を表示します。

ここでは、「開く」ダイアログボックスで次のようなブック「販売記録10.xlsx」を選択した場合の例を示します。なお、Excelの日付・日時データの実体は数値ですが、この判定では数値セルとは見なされません。また、セルC14には全店舗の販売数の合計を求める数式が入力されており、その結果として数値が表示されています。このセルも、数式が文字列と判定されるため、やはり数値にはカウントされません。

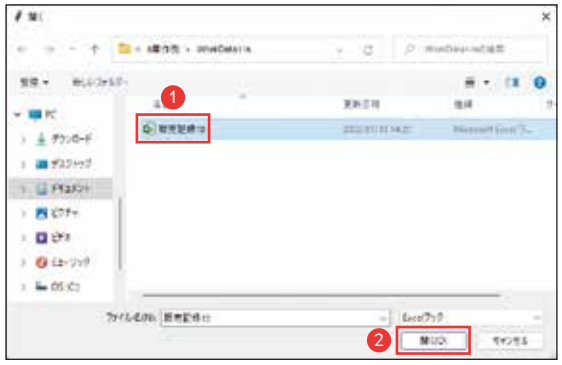
ブック「販売記録10.xlsx」

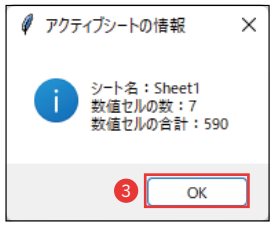


PROGRAM ▸ sample116_1.py

```
import tkinter
from tkinter import filedialog as fd
from tkinter import messagebox as mb
import openpyxl
root = tkinter.Tk()
root.withdraw()
ftype = [('Excelブック', '*.xlsx *.xlsm')]
fname = fd.askopenfilename(filetypes=ftype)
if fname:
    wb = openpyxl.load_workbook(fname)
    ws = wb.active
    total = 0
    ncount = 0
    for row in ws.iter_rows():
        for cell in row:
            if isinstance(cell.value, (int, float)):
                ncount += 1
                total += cell.value
    mb.showinfo('アクティブシートの情報',
                f'シート名:{ws.title}¥n'
                f'数値セルの数:{ncount}¥n'
                f'数値セルの合計:{total}')
```

実行例





Pythonで独自のユーザーインターフェースを構築したい場合は、標準ライブラリの「tkinter」を使用すると便利です。ここでは、ファイルダイアログとメッセージボックスの表示に、tkinterにあらかじめ用意されているダイアログボックスを使用しています。