1.3 ノンプログラマーがPythonを選ぶべき理由

1.3.1 絶大な人気を誇るプログラミング言語Python

さて、数あるプログラミング言語の中で、なぜPythonを選ぶのかについて考えていきましょう。

プログラミング言語Pythonが生まれたのは1990年初頭。意外にも、もうすぐ30歳を迎えるほどの歴史のある言語です。オランダ出身、アメリカ在住のグイド・ヴァンロッサムが、自らも開発に携わっていた教育用のプログラミング言語「ABC言語」の派生としてPythonを開発したとされています。

Pythonは、もともとはグイド・ヴァンロッサムが愛していたコメディ番組「空飛ぶモンティ・パイソン」がその名前の由来となっていますが、「Python」（つまり日本語では「ニシキヘビ」）がそのアイコンとして使用されています。

そんな長い歴史を持つプログラミング言語Pythonですが、本職プログラマーにとっては、数々のプログラミング言語の中で1、2を争うほど人気の言語となっています。

米国の電気工学技術の学会誌「IEEE Spectrum」が発表した人気プログラミング言語の調査「Top Programming Languages 2020[注3)]」では、Pythonが1位となっています。また、「日経xTECH」による「プログラミング言語実態調査（2019）[注4)]」では、PythonはC/C++に次いで2位にランクインしました。つまり、日本国内での人気も高いということがわかります。

なぜ、これほどまでに人気があるのでしょうか？

Pythonの公式サイト「Python.org[注5)]」のトップページ（**図1-3**）には、Pythonについてこのように説明されています。

注3） Top Programming Languages 2020 https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2020
注4） プログラミング言語実態調査 2018　習得したいプログラミング言語、したくない言語 https://tech.nikkeibp.co.jp/atcl/nxt/column/18/00501/111200004/
注5） Welcome to Python.org https://www.python.org/

> Python is a programming language that lets you work quickly and integrate systems more effectively.

直訳的には「Pythonは素早く作業でき、システムをより効果的に統合できるプログラミング言語」となります。

図1-3 Python.orgのトップページ

この一文の意味は、Pythonの4つの特徴を確認することで、より深く理解することができます。そして、そのメリットはノンプログラマーにとっても、有益なものだということを確認していきましょう。

なお、この節ではいくつかのPythonのコードが登場しますが、現時点でその内容を理解する必要はありません。Pythonの世界に触れていただくことが目的なので、ぜひ堪能してみてください。

1.3.2 文法がシンプルである

Pythonの最大の特徴は「文法がシンプルである」ということです。

Pythonプログラミングの基本の考え方がまとめられている「The Zen of Python」（Pythonの禅）[注6)]の一節には、「Simple is better than complex.（単純なものは複雑なものより優れている）」と書かれています。また、ストレートに「Readability counts.（読みやすさが重要）」という一節もあります。

たとえば、「予約語」について見てみると、いかにPythonがシンプルさを重視しているかがわかり

注6） 「The Zen of Python」は、「import this」というコードの実行で確認することができます。Pythonを実行できるようになったら試してみてください。

2.1 Anacondaを準備する

2.1.1 ディストリビューションAnacondaとは

ディストリビューションというのは、Pythonの「本体」のほか、さまざまなモジュールや開発に使用するソフトウェアなどをセットにした配布物のことです。ディストリビューションをダウンロードし、インストールすることで、すぐにPythonが使用できるようになるというのは「バッテリー同梱」にて触れたとおりです。

Pythonのディストリビューションは、いくつかの種類があります。いわゆる「公式」としてPythonソフトウェア財団が「Python.org[注1)]」で提供しているディストリビューションもありますが、今人気が高いディストリビューションは、Anaconda社製の「Anaconda」(アナコンダ) です。

Anacondaはデータサイエンス向けのディストリビューションとうたっていますが、それだけには留まらない豊富なライブラリとJupyter Notebookなどのツール群が同梱されています。ノンプログラマーにとっては、Anacondaの内容で事足りることも多く、追加でライブラリやソフトウェアをインストールする必要がないというメリットがあります。

2.1.2 Anacondaをインストールする

では、Anacondaのインストールを進めていきましょう。

まず、以下URLからAnacondaの個人向けのダウンロードページにアクセスしてください。

Anaconda | Individual Edition
https://www.anaconda.com/products/individual

「Anaconda | Individual Edition」というページが開きますので、[Download] をクリックします。

注1) Python.org https://www.python.org/

図2-1 Anacondaの個人用をダウンロード

MEMO

無償のIndivisual Editionは200名以上の営利組織でビジネス利用する場合は使用することができません。月額$14.95のCommercial Editionを使う必要がありますのでご注意ください。
もしくは、簡易版のMinicondaを使用するという選択肢があります。

ページがスクロールしますので、お使いのOSに合わせたインストーラを選択してクリックしてダウンロードします。「Graphical Installer」がGUIによるインストールで、MacOSではコマンドでインストールをする「Command Line Installer」も提供されています。

本書ではWindowsの64bit版を例に進めていきますが、MacのGUIによるインストールもほぼ同様です。

図2-2 インストーラを選択する

クリックすると、「Anaconda3-20YY.MM-Windows-x86_64.exe」がダウンロードされます。

3.1 Pythonの基本

3.1.1 ステートメントとインデント

Pythonのプログラムを実行すると、記述されたコードの命令が、1つずつインタプリタで解釈、処理されていきます。その命令の最小の単位を「ステートメント（statement）」といいます。ステートメントは日本語では「文」という意味ですね。

ステートメントにはその実行したい内容に応じてさまざまな種類が用意されていて、その種類に応じて記述のルール、つまり構文が決まっています。ここでは、ステートメントの書き方についていくつかの例を用いて解説します。ステートメントの意味については、ここで理解している必要はありませんので、書き方に注目して読み進めてください。

たとえば、「Hello Python!」と出力表示するステートメントは、sample03_01.pyのように記述します。このように1行で記述できるステートメントを「単純文」といいます[注1)]。

sample03_01.py 単純文

```
print('Hello Python!')
```

■ 実行結果

```
Hello Python!
```

単純文で、sample03_02.pyのようにかっこ「()」「[]」「{}」で囲まれた範囲であれば、改行をして記述できます[注2)]。

sample03_02.py 単純文の改行

```
numbers = [
    1, 2,
    3, 4
]
print(numbers)
```

注1） 単純文は、基本的に1行に1文を記述しますが、複数の単純文をセミコロン(;)で区切って1行に記述することができます。
注2） または、バックスラッシュ「\」を入れた位置にも改行を挿入することができます。

■ 実行結果

```
[1, 2, 3, 4]
```

この例ではあまり効果的ではありませんが、ステートメントが長くなりすぎる場合などに、改行を入れて「縦に揃える」ことを意識すると読みやすいコードになります。

また、ステートメントとステートメントの間の空行については、自由に入れることができます。sample03_03.pyのようにステートメントの間に複数の空行を入れても問題なく実行されます。

sample03_03.py 空行

```
print('Hello Python!')

print('Hello World!')
```

■ 実行結果

```
Hello Python!
Hello World!
```

適宜空行を入れることで、処理のまとまりを作ったり、見やすくしたりできます。うまく活用するとよいでしょう。

一方で、一部のステートメントは、他のステートメントを含む形で、複数行で構成します。これを「複合文」といいます。sample03_04.pyの「if」で始まる行から最後までは、1つのステートメントで複合文です。

sample03_04.py 複合文

```
x=10
y=5
if x > y:
    print('xはyより大きい')
elif x == y:
    print('xとyは等しい')
else:
    print('xはyより小さい')
```

■ 実行結果

```
xはyより大きい
```

複合文は1つ以上の「節（clause）」で構成されます。sample03_04.pyであればif節、elif節、else節の3つの節で構成されています。

4.1 if文による条件分岐

4.1.1 if文と条件式

Pythonのプログラムを実行すると、上から順に1つひとつのステートメントが処理されていきます。しかし、実際にプログラムを作成するときには、以下のような条件によって、後の処理を分岐させたいことがあります。

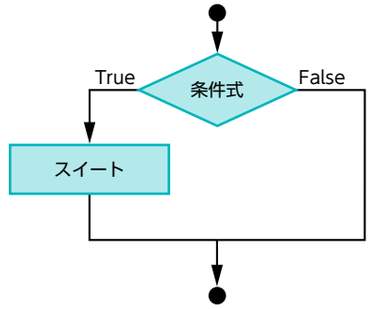
- 変数の値が50以上かどうか
- 今年はうるう年かどうか
- フォルダ内に指定したExcelファイルが存在しているかどうか
- 指定したURLに正常にアクセスできるかどうか

このようなときに、多くのプログラミング言語では、ある条件によって以降の処理を分岐させる仕組みが用意されていて、それを「条件分岐」といいます。Pythonで条件分岐を実現する構文のひとつが「if文」で、以下のように記述します。

```
if 条件式:
    スイート
```

if節の先頭行であるヘッダーにはキーワード「if」に続いて、条件式、そしてコロン記号「:」を記述します。条件式はブール値、すなわちTrueかFalseかいずれかの値を取るものです。条件式が成立している、つまりTrueの値であれば、if節に含まれるスイートを実行し、成立していないとき、つまりFalseであればスイートは実行されずに無視されます。この流れを図に表すと、**図4-1**のようになります。

図4-1 if文による条件分岐



if文の例として、sample04_01.pyを見てみましょう。

sample04_01.py if文による条件分岐

```
1  x = 5
2  y = 3
3
4  if x > y:
5      print('xはyより大きい')
```

■ 実行結果

```
xはyより大きい
```

変数xの値のほうが、変数yの値よりも大きいので、条件式「x > y」は成立している、つまりTrueですから、スイートの処理が実行されます。変数xの値を、変数y以下の値に設定すると、スイートは処理されずに、何も出力されなくなります。試してみてください。

なお、スイートは複数のステートメントで構成することもできますが、スイートに含まれるすべてのステートメントにはインデントを入れる必要があります。

4.1.2 if～else文

if文で条件式がfalseとなった際に、別のスイートを実行したいときがあります。その場合には、if文にelse（エルス）節を加えた、「if～else文」を使うことで実現できます。if～else文は、以下のように、if文に加えてelse節を加えて記述します。

5.1 文字列

5.1.1 文字列とリテラル

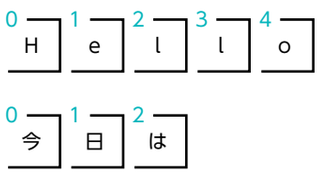
Pythonでプログラムを作るのであれば「文字列」は必ず操作するといってもよいデータです。ExcelやCSVに記録されているデータや、Webからスクレイピングで収集する対象の多くも、ファイル名やドライブ内のファイルの場所を表すファイルパスも文字列です。

さて、Pythonでは、文字列を「文字の集合」として扱います。そのデータ型は「str型」です。つまり、言い換えると、文字列はstr型のオブジェクトです。

文字列は、データを格納する囲いが並んでいるような構造になっています。そこに、文字列の先頭から順番に1文字ずつが格納されているのです。なお、囲いには0からはじまる整数がラベリングされています。

たとえば、「Hello」「今日は」という文字列であれば、それぞれ**図5-1**に示すようなイメージで表現できます。

図5-1 文字列



囲いに付番されている整数を「インデックス（index）」といいます。日本語では「索引」を意味しますね。インデックスは、常に0から順番に付番されています。このインデックスがあることで、文字列に含まれる文字の集合の順番が常に定められています。このように、順番のある要素の集合を「シーケンス（sequence）[注1)]」といいます。日本語では「連続」という意味になります。

Pythonで文字列を直接記述するには、シングルクォート「'」またはダブルクォート「"」で囲みます。

注1） 他のシーケンスとしてリスト、タプル、rangeなどがあります。

これが、「文字列リテラル」です。基本的にはどちらの記号で囲ってもよいのですが、可読性を高めるために、どちらかを一貫して使用するほうがよいでしょう。

MEMO

本書では、とくに理由がない限り、文字列リテラルにシングルクォートを使用します。

また、たとえばsample05_01.pyのように、文字列の中でシングルクォート（またはダブルクォート）を使用するのであれば、他方のクォート記号で囲むのがひとつの手段となります。

sample05_01.py 文字列リテラル

```
1  print('今日は')
2
3  msg = "Hello! I'm fine."
4  print(msg)
5
6  print("")
```

■実行結果

```
1  今日は
2  Hello! I'm fine.
```

なお、「''」や「""」というように、シングルクォートまたはダブルクォートを連続して2つ続けることで「0文字の文字列」を表現できます。このような、文字要素を1つも持たない文字列を「空文字列（からもじれつ）」などといいます。

また、複数行にわたる文字列を記述したい場合に、「三重引用符[注2)]」という記述のしかたが用意されています。シングルクォートまたはダブルクォートを3つ連続させた「'''」または「"""」で囲むというものです。sample05_02.pyはその使用例です。

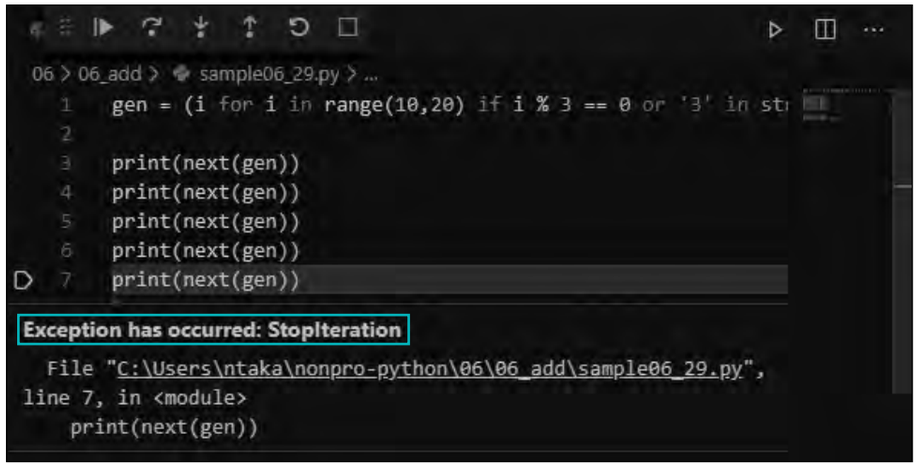
sample05_02.py 三重引用符

```
1  print('''今日は
2  お元気ですか''')
3
4  msg = """Hello!
5  How are you?
6  I'm fine."""
7  print(msg)
```

注2） 三重引用符は、関数などの仕様を記述するためのドキュメンテーション文字列にも使用されます。これについては6章で解説します。

結果を見ると、next関数で順番にその要素を取り出すことができています。また、コメントアウトをしているステートメントについてコメント化を外して実行すると、図6-7のように「StopIteration」が発生します。これらの性質から、ジェネレータ式で生成されたオブジェクトがイテレータであることがわかります。

図6-7 例外StopIteration



一方で、イテレータについてtype関数でその型を確認すると、この例のとおり「<class 'str_iterator'>」などと表示され、イテレータであることがわかります。しかし、ジェネレータ式で生成したオブジェクトは「<class 'generator'>」と表示されますので、これを見て、イテレータではないと思われるかもしれません。

しかし、ジェネレータ式で生成されるのは、「ジェネレータイテレータ（generator iterator）」というイテレータの一種です。ジェネレータ式またはジェネレータ関数[注5)]で生成されたイテレータをそのように呼びます。型は違えどもイテレータですので、これまで紹介してきたイテレータと同様の性質を持ちます。

このように、Pythonでは内包表記やジェネレータ式を用いることで、リスト、辞書、イテレータといったオブジェクトを、シンプルな記法で生成できるのです。

注5） ジェネレータ関数はreturn文の代わりにyield文で戻り値を設定します。通常の関数では関数の戻り値がreturn文で指定したオブジェクトになります。yield文のあるジェネレータ関数では、その戻り値がジェネレータイテレータとなります。

6.4 クラス

6.4.1 クラスとは

たとえば、商品ID、商品名、価格を持つ商品のデータをPythonで取り扱いたいとします。sample06_31.pyのような、商品情報を出力表示するプログラムを作りました。

sample06_31.py 商品情報を出力する

```
rice_balls = [
    (1, 'Sake', 110),
    (2, 'Tuna', 120),
    (3, 'Ume', 100)
]

for rice_ball in rice_balls:
    id = rice_ball[0]
    name = rice_ball[1]
    price = rice_ball[2]
    print(id, f'{name} の価格は {price} 円です')
```

■ 実行結果

```
1 Sake の価格は 110 円です
2 Tuna の価格は 120 円です
3 Ume の価格は 100 円です
```

各商品のデータをタプルで表現していて、そのタプルに対してインデックスを用いて商品ID、商品名、価格を取り出しています。わかりやすさのために、変数id、変数name、変数priceという名前の変数を用意してそこにいったん代入をしています。

これでもプログラムとしてはまったく間違いではありません。ですが、sample06_31.pyのfor文について、以下のように書くことができるとしたらどうでしょうか？

sample06_32.py 商品のデータを取り出す

```
for rice_ball in rice_balls:
    print(rice_ball.id, f'{rice_ball.name} の価格は {rice_ball.price} 円です')
```

7.1 Excelにデータを集めるツールの概要

7.1.1 Pythonでデータを集める

「データを集める」つまり、複数のファイルから、データを抽出して1つのファイルにまとめるという作業は多くの業務で発生します。

たとえば、よくその対象となるファイル形式の筆頭が「xlsx形式」です。主に、表計算ソフトExcelにて使用される形式で、そのファイルの拡張子は「.xlsx」になります。「Excel形式」などとも呼ばれますね。多くの職場でExcelを使用していますから、当然その操作する頻度も高いといえます。

もうひとつ、よく使用されるのが「csv形式」のファイルです。「csv」とは「comma-separated values」の略で、カンマ区切りでデータを格納する形式です。単なるテキストデータなので、ファイル容量も小さく、さまざまな環境で取り扱いが可能であるということから、システム間のデータの受け渡しによく使用されます。

ここで、複数のcsvファイルのデータを、xlsxファイルにまとめていく作業について考えてみましょう。

xlsxファイルだけならExcelの基本機能、または、それに加えてVBAというプログラミング言語で作業を行うのがひとつの選択肢となりえます。しかし、csvファイルを扱うときに「文字化け」の問題が起きてしまうことがあります。

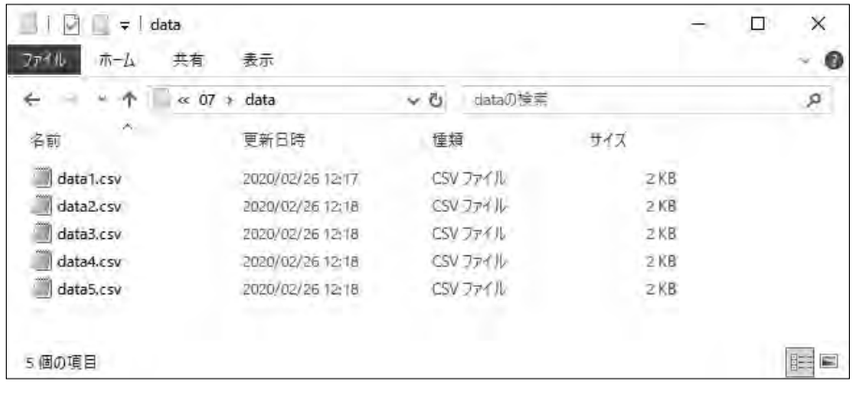
Pythonであれば、データ処理を得意とするライブラリ「pandas（パンダス）」を使って、それらの課題をスマートに解決できます。pandasはcsvファイルも、xlsxファイルも両方取扱うことができ、文字化けの問題にもスマートに対処できるのです。また、フォルダ内の複数のファイル操作については、組み込みモジュールの「pathlib（パスリブ）」を使うことができます。

このように、Pythonではライブラリが豊富に提供されているので、作業したいそれぞれの領域について、その操作を得意とするライブラリ見つけやすく、それを組み合わせることで希望のツールを完結させることができる。つまり、弱点が少ないわけです。それがPythonを学ぶ大きなメリットのひとつです。

7.1.2 データを集めるツール

この章で題材となる、データを集めるツールの概要をお伝えしておきましょう。まず、**図7-1**のようにフォルダ「data」の中に複数のcsvファイルが保管されています。

図7-1 csvファイルが格納されたフォルダ



フォルダの構成は、以下のようにプロジェクトフォルダ「nonpro-python」→章を表すフォルダ「07」→データフォルダ「data」としています。

```
nonpro-python
└07
  └data
    └data1.csv
    └data2.csv
    └data3.csv
    └data4.csv
    └data5.csv
```

これらのcsvファイルはすべて同じフォーマットで、その内容は**図7-2**のようになっています。

8.1

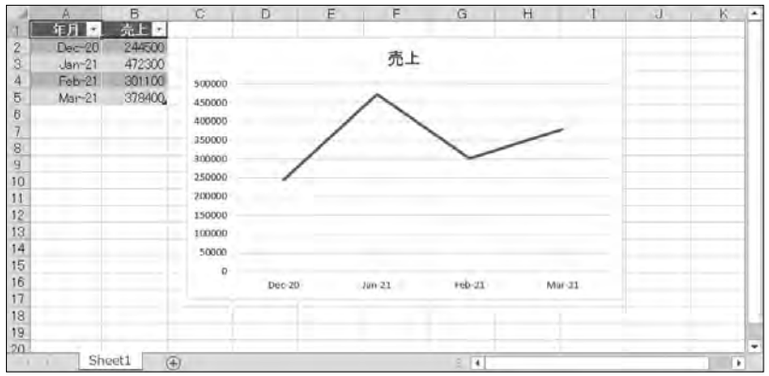
Excelレポートを更新するツールの概要

8.1.1 PythonによるExcelファイルの更新

売上や販売数などの日々のデータをExcelレポートに追加・更新という業務は多く存在しています。Excelファイルへのデータ書き込みはpandasのto_excelメソッドでも可能ですが、その方法では、もともとからExcelファイルに含まれている書式設定やグラフなどを保持することができません。

たとえば、図8-1のようなExcelファイルがあるとします。

図8-1 テーブルとグラフを含むExcelファイル



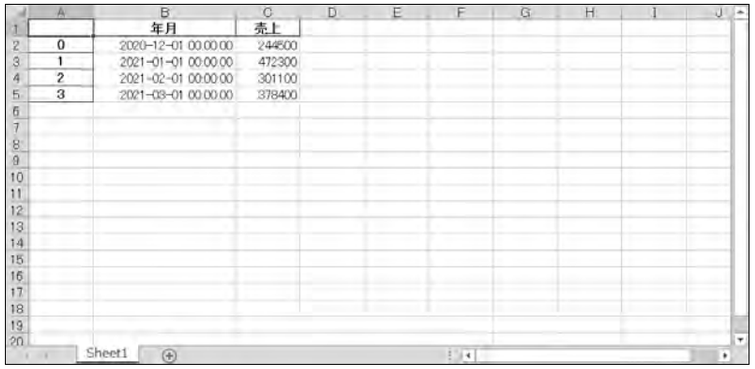
これに対してsample08_01.pyを実行してみましょう。シンプルに「08\sample.xlsx」のデータを読み取り、同じファイルに書き込むというだけのプログラムです。

sample08_01.py pandasによるExcelファイルの更新

```
1  import pandas as pd
2
3  filename = r'08\sample.xlsx'
4  df = pd.read_excel(filename)
5  print(df.head())
6  df.to_excel(filename)
```


実行後、「08\sample.xlsx」を開いたものが図8-2のようになります。つまり、pandasによる書き込みの際に、テーブルや書式設定、グラフが失われてしまうのです。

図8-2 pandasによる書き込み後のExcelファイル



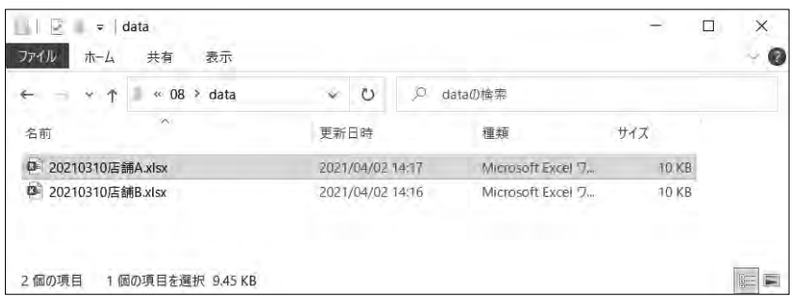
このように、pandasはデータそのもの以外についての操作、つまりExcelファイルの書式設定、テーブル、グラフなどの操作があまり得意ではありません。

書式設定やグラフなどを保持したり、操作したりする場合は、別のライブラリである「openpyxl（オープンパイエクセル、またはオープンパイエックスエル）」を使うことができます。ここでは、openpyxlを用いてExcelファイルの操作をする方法について学んでいきましょう。

8.1.2 レポートを更新するツール

では、今回目標とするツールについて確認していきましょう。まず、図8-3のようにフォルダ「data」の中に複数のExcelファイルが格納されています。店舗が複数あって、それぞれから日別の売上データがExcelファイルとして提出されたものを格納しているというイメージです。

図8-3 Excelファイルが格納されたフォルダ



```
 7
 8  def openWorkbook():
 9      wb = load_workbook(filename)
10      ws = wb.active
11      return (wb, ws)
12
13  def closeWorkbook(wb):
14      wb.save(filename)
15      variable.set('')
16
17  def begin():
18      now = datetime.now()
19      wb, ws = openWorkbook()
20      ws.append([now.date(), now.time(), variable.get()])
21      button_begin['state'] = tk.DISABLED
22      button_finish['state'] = tk.NORMAL
23      closeWorkbook(wb)
24
25  def finish():
26      now = datetime.now()
27      wb, ws = openWorkbook()
28      max_row = ws.max_row
29      ws.cell(max_row, 4).value = now.time()
30      ws.cell(max_row, 5).value = variable.get()
31      button_begin['state'] = tk.NORMAL
32      button_finish['state'] = tk.DISABLED
33      closeWorkbook(wb)
34
35  root = tk.Tk()
36  root.title('出退勤ツール')
37
38  button_begin = ttk.Button(text='出勤', command=begin)
39  button_begin.pack(side='left')
40
41  button_finish = ttk.Button(text='退勤', command=finish, state=tk.DISABLED)
42  button_finish.pack(side='left')
43
44  variable = tk.StringVar()
45  entry = ttk.Entry(textvariable=variable)
46  entry.pack(side='left')
47
48  root.mainloop()
```

このコードで使用しているtkinterパッケージ、datetimeモジュールとその使い方について以降で学んでいくことにしましょう。

9.2 インターフェースを作る：tkinter

9.2.1 tkinterパッケージ

ウィンドウ、ボタン、テキストボックスなどといった部品を組み合わせたインターフェースを作るためのパッケージが「tkinter[注1)]」です。

tkinterパッケージにはいくつかのモジュールが含まれていますが、その中心となるのは「tkinter」モジュールです。また、その配下に「ttk」や「messagebox」といったいくつかのモジュールを含んでいます。これらのように、あるモジュールの配下に位置するモジュールを「サブモジュール」といいます。tkinterパッケージに含まれるモジュールの一部について、**表9-1**にまとめていますのでご覧ください。

表9-1 tkinterパッケージに含まれるモジュールの一部

モジュール	説明
tkinter	インターフェースを作成する基本のモジュール
ttk	テーマ付きウィジェットの機能を提供するモジュール
messagebox	メッセージダイアログを使用する機能を提供するモジュール
filedialog	ファイルダイアログを使用する機能を提供するモジュール
colorchooser	色選択ダイアログを使用する機能を提供するモジュール

独自のインターフェースを作成するのであれば、tkinterモジュールとttkモジュールを主に使用します。これら2つのモジュールにより、さまざまな種類のウィジェットを部品としてインターフェースに配置し、操作をすることができます。

また、OS等であらかじめ用意されているダイアログを操作するのであれば、messageboxモジュール、filedialogモジュール、colorchooserモジュールといったモジュールを使用します。

tkinterパッケージはPythonに組み込まれていますので、これらのモジュールのうち、必要なモジュールをインポートすることで使用できます。

注1） 公式ドキュメントページ: https://docs.python.org/ja/3/library/tkinter.html

の方法を紹介します。

Webページの表を取得するには、pandasのread_html関数を使います。構文は以下のとおりです。

```
pandas.read_html(io, header=None, index_col=None, encoding=None)
```

パラメータioに、対象とするWebページのURLを指定します。これにより、そのページ上にある表をデータフレームとして取得します。ページ上には複数の表が存在し得るため、read_html関数の戻り値は、データフレームのリストとなります。

パラメータheaderおよびindex_colは、それぞれカラムとして使用する行またはインデックスとして使用する列表す整数を指定します。パラメータencodingには使用する文字コードを指定します。これらのパラメータはread_csv関数のものと同様ですね。

では、使用例としてsample10_15.pyをご覧ください。

sample10_15.py pandasによる表のスクレイピング

```
1  import pandas as pd
2
3  url = 'https://tonari-it.com/scraping-test/'
4  dfs = pd.read_html(url)
5  print(dfs[0])
```

■ 実行結果

```
1          メンバー                                説明
2  0       find           タグ名、属性名等で要素を抽出
3  1  find_all      タグ名、属性名等で要素のリストを抽出
4  2     select  CSSセレクタselectorで要素を抽出
```

とても簡単に表データの取得ができますね。ぜひご活用ください。

10.4 スクレイピングツールを作る

10.4.1 検索結果ページのHTMLドキュメントを取得する

では、「いつも隣にITのお仕事」の検索結果ページから記事タイトルとURLを取得するスクレイピングツールを作成していきましょう。このサイトの検索結果ページは、検索キーワードを「query」とすると、以下のURLでアクセスできます。

```
https://tonari-it.com/?s=query
```

ですから、まずはrequestsを使ってこのURLにリクエストを行い、そのレスポンスからHTMLドキュメントを取り出すのが最初のステップとなります。sample10_02.pyとほぼ同様のコードになりますが、改めてsample10_16.pyとして作成しました。

実行して、取得したHTMLドキュメントのtitleタグが『「Python 初心者」の検索結果 | いつも隣にITのお仕事』となっていることを確認しておきましょう。

sample10_16.py requestsによるHTMLドキュメントの取得

```
1  import requests
2
3  url = 'https://tonari-it.com'
4  query = 'Python 初心者'
5  params = {'s': query}
6  r = requests.get(url, params=params)
7  r.raise_for_status()
8
9  print(r.text[0:3000])
```

■ 実行結果

```
1  <!doctype html>
2  <html lang="ja"
3          prefix="og: https://ogp.me/ns#" >
4
5  <head>
6  <meta charset="utf-8">
```