Pythonを用いたデータ蓄積方法

ファイル入出力

`open()` ファイルを開く

```
f = open("ファイル名", "読み込みモード", encoding="テキストエンコーディング(default:utf-8)")
```

読み込みモードについて

- `'r'` 読み込み用に開く（デフォルト）
- `'w'` 書き込み用に開く（新規作成・上書き）
- `'x'` 排他的な生成に開き，ファイルが存在する場合は失敗する
- `'a'` 書き込み用に開き，ファイルが存在する場合は末尾に追記する
- `'b'` バイナリモード（バイナリ読み込みの場合 `'rb'` と指定する）
- `'t'` テキストモード（デフォルト）

変数 `f` にはファイルを読み書きするためのデータが入り，fileオブジェクトと呼びます．

fileオブジェクトのメソッドは下記の通りです.

- `close()` ファイルを閉じる
- `read()` ファイル全体を読み込む
- `readlines()` ファイルを一行づつ読み込んで行をリストで返す
- `write()` ファイルを書き込む

`write()` のサンプルコード

```
csvtext = 'a,b,c\n1,2,3\n'

f = open("text.csv", "w", encoding="shift_jis")
f.write(csvtext)
f.close()
```

※文字列を定義する際に `\n` を指定すると改行コードの `LF`, `\r` の場合は改行 `CR` が定義される.

`read()` と `readlines()` のサンプルコード

```
csvtext = ''
csvlines = []

f = open("text.csv", "r", encoding="shift_jis")
csvtext = f.read()
f.close()

f = open("text.csv", "r", encoding="shift_jis")
csvlines = f.readlines()
f.close()

print(csvtext)
print(csvlines)
```

`read()` はファイル全体を文字列として読み込みを行うのに対し，`readlines()` は（テキストモードの場合）ファイルの各行を文字列のリストとして読み込みます.

モジュールを用いたCSVファイルの読み書き

モジュールのインポート

モジュールをインポートするには，モジュールの関数やメソッド，クラスが呼ばれる前に `import` を用います.

```
import [モジュール名]
```

数学関係の機能をまとめた `math` モジュールを利用したい場合のサンプルコード：

```
import math                    # importは大抵セルの一番上に記述します
print(math.sqrt(2))            # sqrt は平方根を計算する関数
print(math.pi)                 # πの値
print(math.sin(math.pi/4))     # sin関数（引数の角度はラジアン表記）
```

モジュールを利用する場合，`pip` 等のコマンドを事前に実行してインストールを行っておきます.

csvモジュールを用いたCSVファイルの読み込み

```
import csv
f = open('text.csv', 'r')
dataReader = csv.reader(f)
for row in dataReader:
    print(row)
f.close()
```

pandasを利用する場合

```
import pandas as pd

df = pd.read_csv('text.csv')
print(df)
```

その他，数値計算関連のモジュールnumpyなどにもCSVを取り込むメソッドが用意されている．

練習問題

下記のCSVファイルがある．２列目の分類が`1`である項目のみ，４列目の購入額の合計を計算するとともに，２列目と４列目のみを抽出して新たなCSVファイルを書き出ししたい．

ID	分類	品名	購入額	購入日
1	1	ノートPC	250000	2021/11/5
2	1	ノートPC	210000	2021/11/15
3	2	Microsoft Windows10	16000	2021/12/2
4	2	Microsoft Office	34000	2021/12/4
5	1	デスクトップPC	130000	2021/12/20
6	3	モバイルバッテリー	5200	2022/1/20

1. CSVファイルを読み込む
2. CSVファイルを読み込みながら書き出す要素を取り出す
3. 書き出す要素をCSVファイルに書き出す

設問のCSVを書き出すサンプルコード

```
if __name__=='__main__':
  csvdata = (("ID","分類","品名","購入額","購入日"),
  ("1","1","ノートPC","250000","2021/11/5"),
  ("2","1","ノートPC","210000","2021/11/15"),
  ("3","2","Microsoft Windows10","16000","2021/12/2"),
  ("4","2","Microsoft Office","34000","2021/12/4"),
  ("5","1","デスクトップPC","130000","2021/12/20"),
  ("6","3","モバイルバッテリー","5200","2022/1/20"))

  f = open("sample.csv", "w", encoding="shift_jis")
  for item in csvdata:
      f.write(",".join(item))
      f.write("\n")
  f.close()
```

解答のサンプルコード

【補足】pythonのキーワード

キーワードは変数名や関数名，クラス名などの名前（識別子）として使えないもののことを指します．コード内に記載すると言語仕様上特別な意味を持った語が該当します.

```
import keyword
print(keyword.kwlist)
```

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

【補足】キーワード以外に使えないもの

キーワードの他にも，変数名や関数名，クラス名などの名前（識別子）として宣言・定義を行うと意図しないエラーに直面する事例がいくつかあります.

1. ソースファイル名とライブラリ名が重複している
2. 自身で定義した変数・関数・クラス名と組み込み関数名が重複している
3. 識別子に大文字小文字のアルファベット（`A~Z`，`a~z`）と数字（`0~9`）、アンダースコア（`_`）以外のASCII文字を使っている

例）組み込み関数と自身で宣言した変数名が重複した例

```
len = 0
numbers = ['a', 'b', 'c']

print(len)
print(len(numbers))
```

Excelファイルの取り扱い

`openpyxl` や `pandas` を用いてExcelファイル（xls, xlsx）も扱うことができる.

```
import openpyxl
import datetime
if __name__=='__main__':
    workbook = openpyxl.load_workbook('reservationTemplate.xlsx')      # テンプレートファイルを読み込む
    # workbook = openpyxl.Workbook() # 新規でExcelファイルを作る場合

    worksheet = workbook.get_sheet_by_name(u'予約') #シートを選択する
    worksheet.cell(row=5, column=6).value = unicode('12345678', 'utf-8')    # セルに値を代入する
    # 行、列は1開始のインデックスなので注意

    currentFormat = worksheet['AA5'].number_format # セルの書式フォーマットを設定する

    worksheet['AA5'].value = datetime.date.today() # datetimeモジュールを用いて現在日時を取得して入力
    worksheet['AA5'].number_format = currentFormat # セルの書式を設定する
    workbook.save('reservation.xlsx')    # 編集したブックを保存する
```

データベースとの連携

データベース

> [データベースとは | Oracle 日本]
>
> データベースとは、**構造化した情報またはデータの組織的な集合**であり、通常はコンピューター・システムに電子的に格納されています。 データベースは通常、データベース管理システム（DBMS）で制御します。 データとDBMS、およびそれらに関連するアプリケーションをまとめてデータベース・システムと呼びます。 ”

データベースの種類

- **リレーショナル・データベース**：リレーショナル・データベースは1980年代に主流となりました。リレーショナル・データベースの各項目は、列と行を持つテーブルの集合として編成されています。リレーショナル・データベースのテクノロジーは、構造化された情報にアクセスする最も効率的で柔軟な方法を提供します。
- **オブジェクト指向データベース**：オブジェクト指向データベースに保存した情報は、オブジェクト指向プログラミングの場合と同様に、オブジェクトとして扱われます。
- **NoSQLデータベース**：NoSQLデータベース（非リレーショナル・データベース）では、非構造化データと半構造化データを格納し、操作できます（これとは対照的に、リレーショナル・データベースではデータベースに挿入するすべてのデータの構成方法が規定されています）。Webアプリケーションが広く使用され、複雑になるに伴い、NoSQLデータベースが普及するようになりました。

> cf.) データベースとは | Oracle 日本 ”

データベース管理システム（DBMS）とは

> 多くの場合、データベースには、データベースの包括的なソフトウェア・プログラムであるデータベース管理システム（DBMS）が必要です。DBMSは、データベースとそのエンド・ユーザーまたはプログラムとの間でインターフェイスとして機能し、情報の編成方法と最適化方法をユーザーが取得、更新、および管理できるようにします。また、DBMSはデータベースの監視と制御を容易にし、パフォーマンスの監視、チューニング、バックアップとリカバリなどのさまざまな管理操作を可能にします。”

広く使用されているデータベース・ソフトウェア（DBMS）の例として、MySQL、Microsoft Access、Microsoft SQL Server、FileMaker Pro、Oracle Database、dBASEがあります。（※）

> cf.) データベースとは | Oracle 日本 ”

※オープン系ではPostgreSQL，DB2などもある

pythonとDBMSの連携を行うモジュール例

- MySQL
  - mysql-connector-python
  - PyMySQL
- SQLite
  - sqlite3
- Microsoft Access
  - pyodbc
- Microsoft SQL Server (Microsoft| Python SQL ドライバー)
  - pyodbc
  - pymssql
- Oracle Database
  - cx_oracle

リレーショナルデータベース（RDB）について

現在最も広く利用されているデータベースですで，Excelシートを例に取ると，列と行からなる表の形式でデータを管理するもので，SQL（Structured Query Language）を用いて操作を行うことが一般的です．RDBを扱う管理ソフトウェアをRDBMSと呼ぶことがあります．

また，データ構造化する単位をテーブルと呼びます．テーブルにはカラム(Column)が存在し，データはレコードとして記録されます．

※ExcelとRDBにおける用語の比較：
「行：レコード・Row」，「列：カラム(Column)」，「セル：フィールド」

SQLの基礎 (DML:Data Manipulation Language)

- `SELECT` データを取り出す
- `INSERT` データを追加する
- `UPDATE` データを更新する
- `DELETE` データを削除する

SELECT データを取り出す

すべてのデータを取り出す場合の基本的な構文は下記の通り.

```
SELECT [カラム名] from [テーブル名];
```

なお，カラム名に `*` を指定するとテーブル内のすべてのカラムの値を取得することができる.

取り出すデータの条件を指定する場合は `where` 句や条件演算子を用いて指定する. `AND` や `OR` なども利用して条件式を構築することができる. また `ORDER BY` 句により昇順や降順に並び替えて取得することができることに加えて, `LIMIT` 句を用いて取得件数を絞ることも可能.

```
SELECT [カラム名] from [テーブル名] where [カラム名] [比較演算子] [値];
```

ex.) `customer` テーブルから `20 < age < 40` のすべてのカラムのフィールドを取得する

```
SELECT * FROM customer where age > 20 and age < 40 ORDER BY age DESC;
```

`INSERT` データを追加する

```
INSERT INTO [テーブル名] ([列名1], [列名2],...) VALUES ([値1], [値2[],...);
```

テーブルに定義されたカラムの順に値が定義できる場合，カラム名を省略できる場合がある.

```
INSERT INTO [テーブル名] VALUES ([値1], [値2[],...);
```

ex.) `customer` テーブルにレコードを追加する

```
INSERT INTO customer (name, age) VALUES ('Taro Yamada', 35);
```

【応用】サブクエリ（副問い合わせ）を用いてINSERTする内容をSELECTで取得する例.

```
INSERT INTO employees (emp_no, mgr_name) VALUES ( '10106',
    (SELECT emp_name FROM employees WHERE emp_no = '10001'));
```

UPDATE データを更新する

すべてのデータを更新する場合の基本的な構文は下記の通り.

```
UPDATE [テーブル名] SET [カラム名] = [値];
```

条件を指定する場合はSELECT文同様に `where` 句を用いて指定する.

```
UPDATE [テーブル名] SET [カラム名] = [値] where [カラム名] [比較演算子] [値];
```

`DELETE` データを削除する

テーブル上のすべてのデータを削除する構文は下記の通り.

```
DELETE FROM [テーブル名];
```

条件を指定する場合はSELECT，UPDATE文同様に `where` 句を用いて指定する. `ORDER BY` 句を利用して上位または下位から選択する条件も指定可能となっている.

```
DELETE FROM [テーブル名] where [カラム名] [比較演算子] [値];
```

ex.) `users` テーブルのIDを降順に並べて，大きい順に５件のレコードを削除する例

```
DELETE FROM users ORDER BY id DESC LIMIT 5;
```

SQL文の作法

- SQL文の最後に「;」をつける
- 大文字・小文字は区別されない
- 定数の書き方には決まりがある
  - 文字列はシングルクォーテーションで囲む
  - 数値はシングルクォーテーションで囲わない
- 単語は半角スペースか改行で区切る

PythonからSQLiteに接続するサンプルコード

事前に `sqlite3` モジュールのインストールをしておく

```
$ python3 -m pip install sqlite3
```

DBに接続する

`sqlite3.connect()` の引数にDB名（path）を指定して開き，`close()` で切断する．

```
import sqlite3

dbname = 'main.db'
# DBを作成する（既に作成されていたらこのDBに接続する）
conn = sqlite3.connect(dbname)

# DBとの接続を閉じる(必須)
conn.close()
```

SQLを実行する

テーブルの作成

```
import sqlite3

dbname = 'main.db'
# DBを作成する（既に作成されていたらこのDBに接続する）
conn = sqlite3.connect(dbname)

# SQLiteを操作するためのカーソルを作成
cur = conn.cursor()

# テーブルの作成
cur.execute(
    'CREATE TABLE items(id INTEGER PRIMARY KEY AUTOINCREMENT, name STRING, price INTEGER)'
)

# DBとの接続を閉じる(必須)
conn.close()
```

テーブルにデータを登録する

```
import sqlite3

dbname = 'main.db'
# DBを作成する（既に作成されていたらこのDBに接続する）
conn = sqlite3.connect(dbname)
# SQLiteを操作するためのカーソルを作成
cur = conn.cursor()

# データ登録
cur.execute('INSERT INTO items values(0, "りんご", 100)')

# コミットしないと登録が反映されない
conn.commit()

# DBとの接続を閉じる(必須)
conn.close()
```

テーブルにデータを登録する（リストを登録する）

```
import sqlite3

dbname = 'main.db'
# DBを作成する（既に作成されていたらこのDBに接続する）
conn = sqlite3.connect(dbname)
# SQLiteを操作するためのカーソルを作成
cur = conn.cursor()

# データ登録
# 登録するデータ
inserts = [(1, "みかん", 80), (2, "ぶどう", 150), (3, "バナナ", 60)]
# 複数データ登録
cur.executemany('INSERT INTO items values(?, ?, ?)', inserts)

# コミットしないと登録が反映されない
conn.commit()

# DBとの接続を閉じる(必須)
conn.close()
```

テーブルのデータ取得

```
import sqlite3

dbname = 'main.db'
# DBを作成する（既に作成されていたらこのDBに接続する）
conn = sqlite3.connect(dbname)
# SQLiteを操作するためのカーソルを作成
cur = conn.cursor()

# データ取得
cur.execute('SELECT * FROM items')

# 取得したデータはカーソルの中に入る
for row in cur:
    print(row)

# DBとの接続を閉じる(必須)
conn.close()
```

テーブルのデータ更新や削除

`execute()` に更新や削除を行うSQL文を渡すことで反映される.

```
# データ更新
cur.execute('UPDATE items SET price = 260 WHERE id = "3"')
# データ削除
cur.execute('DELETE FROM items WHERE id = "2"')
```

【参考】pyodbcを用いたサンプルコード

Microsoft Access，Microsoft SQL Server等に接続する際は `pyodbc` モジュールを利用する

```
import pyodbc
# Some other example server values are
# server = 'localhost\sqlexpress' # for a named instance
# server = 'myserver,port' # to specify an alternate port
server = 'tcp:myserver.database.windows.net'
database = 'mydb'
username = 'myusername'
password = 'mypassword'
cnxn = pyodbc.connect('DRIVER={ODBC Driver 17 for SQL Server};SERVER='+server
  +';DATABASE='+database
  +';UID='+username
  +';PWD='+ password)
cursor = cnxn.cursor()
```

cf.)pyodbc を使用した SQL への接続を概念実証する

commitとは

トランザクションの処理を確定させる処理のこと.
トランザクションとは、一連の処理をひとまとめにしたもので，途中で中断すると以降の処理に問題が出る場合に分割できない処理を定義するもの.

ex.) トランザクションとして行うべき処理例

1. 利用者が商品を購入したので購入処理開始（トランザクション開始）
2. 「在庫テーブル」で保持している商品の在庫を減らす
3. 「注文テーブル」に購入者情報を登録
4. 処理終了（トランザクション終了）

トランザクションを確定（commit）しない（できない）場合，トランザクションはロールバックが行われ，DBは処理前の状態に保たれる.