2022年12月05日(月) 4限
@研究棟A302

# プログラミングA2
# 第10回

**担当：伏見卓恭**
**連絡先：fushimity@edu.teu.ac.jp**
**居室：研A1201**

# プログラミングA2の流れ

第 1回：＜復習編＞関数，ファイル入出力，コンテナデータ型
第 2回：＜復習編＞クラスとオブジェクト
第 3回：＜文法編＞関数の高度な利用法１
第 4回：＜文法編＞関数の高度な利用法２
第 5回：＜文法編＞オブジェクト指向プログラミング
第 6回：＜応用編＞データ構造とアルゴリズム１
第 7回：＜応用編＞データ構造とアルゴリズム２
第 8回：＜実践編＞HTTPクライアント
第 9回：＜実践編＞スクレイピング
第10回：＜実践編＞データベース
第11回：＜実践編＞並行処理
第12回：＜総合編＞総合演習(複合問題)
第13回：＜総合編＞まとめ
第14回：＜総合編＞ Python力チェック ← 確認テストのこと

# 本日のお品書き

- **データベース**
  - SQLiteへの接続
  - テーブルの作成，削除，更新
  - データの挿入，削除，検索
  - ユーザ定義関数
  - ~~トランザクション~~
  - ~~テーブルの結合~~

# Pythonにおけるデータ保存

- フラットテキストファイル
- 表形式テキストファイル
  - CSV
  - XML
  - JSON
  - Pandas
- リレーショナルデータベース
  - SQLite
  - MySQL
  - PostgreSQL
- **NoSQL**
- バイナリ形式
  - pickle

# sqlite3

標準ライブラリのドキュメント：
https://docs.python.org/ja/3.9/library/sqlite3.html

SQL：https://www.w3schools.com/sql/default.asp

# SQLite

SQLiteは，軽量な関係データベース管理システムである．
サーバではないため設定が不要であり，ライブラリを用いて
アプリケーションに組み込むことができる．


| 商品 ID | 商品名 | 商品分類 | 販売単価 | 仕入単価 | 登録日 |
|---|---|---|---|---|---|
| 0001 | Tシャツ | 衣服 | 1000 | 500 | 2009-09-20 |
| 0002 | 穴あけパンチ | 事務用品 | 500 | 320 | 2009-09-11 |
| 0003 | カッターシャツ | 衣服 | 4000 | 2800 | |
| 0004 | 包丁 | キッチン用品 | 3000 | 2800 | 2009-09-20 |

画像の出典： https://codezine.jp/article/detail/12216

**SQLiteの使い方**

```
import sqlite3

con = sqlite3.connect(データベース名)
cur = con.cursor()

cur.execute(SQL文)

con.close()
```

←DBに接続
←SQLiteを操作するための
　カーソルオブジェクト生成
←SQL文の実行

←DBを切断

# 主要な関数，メソッド

- **connect(データベース名)：**
  - データベースへの接続を確立する
  - ユーザ名，パスワード，サーバアドレスなどを指定可
  - Connectionオブジェクトを返す
- **cursor()：**
  - Connectionクラスのインスタンスメソッド
  - クエリを管理するCursorオブジェクトを生成する
- **execute(SQL文)，executemany(SQL文，データ列)**
  - Cursorクラスのインスタンスメソッド
  - データベースに対して1つor複数のSQLを実行する

    ※講義資料では省略しているが，SQL文実行時は例外が発生しうるので，try-except構文を使うのが一般的
- **fetchone()，fetchmany()，fetchall()**
  - Cursorクラスのインスタンスメソッド
  - execute()の結果を取得する

# テーブルの作成

※本講義資料では，SQLのキーワードは大文字で表記しているが，
小文字で記述しても問題ない．

データ型は必須ではない
↓

**sqlite3によるテーブル作成**

```
cur.execute("CREATE TABLE テーブル名 (カラム名1 データ型1,
                                      カラム名2 データ型2, …)")
```

**create_table.py**

```
import sqlite3

if __name__ == "__main__":
    con = sqlite3.connect("lec10/databases/pokemon.db")
    cur = con.cursor()
    cur.execute("CREATE TABLE names (id INTEGER, name TEXT, types TEXT)")
    con.close()
```

4カラムの表を作成
1. id（整数）
2. name（文字列）
3. types（文字列）
↓

※同じ名前のテーブルを複数回作成できない ↓

**実行例**

```
PS C:¥Users¥admin¥Desktop¥ProA2> python ./lec10/create_table.py
Traceback (most recent call last):
    cur.execute("CREATE TABLE names (id INTEGER, name TEXT, types TEXT)")
sqlite3.OperationalError: table names already exists
```

【練習】ためしに，同じ名前のテーブルを作ってみよう．

# 練習問題：create_table2.py

6つの種族値(hp, atk, dfs, sp_atk, sp_dfs, spd)を格納するテーブルstatsをデータベースpokemon.dbに作成せよ.

create_table2.py

```
import sqlite3

if __name__ == "__main__":
    con = sqlite3.connect("lec10/databases/pokemon.db")
    cur = con.cursor()
    cur.
                                                    )
    con.close()
```

# テーブルの削除

**sqlite3によるテーブル削除**

```
cur.execute("DROP TABLE テーブル名")
```

**create_table.py**

```
import sqlite3

if __name__ == "__main__":
    con = sqlite3.connect("lec10/databases/pokemon.db")
    cur = con.cursor()
    cur.execute("DROP TABLE names")
    con.close()
```

※存在しないテーブルは削除できない ↓

**実行例**

```
PS C:¥Users¥admin¥Desktop¥ProA2> python ./lec10/create_table.py
Traceback (most recent call last):
    cur.execute("DROP TABLE names")
sqlite3.OperationalError: no such table: names
```

【練習】テーブル作成後，テーブルを削除してみよう．

# カラムの追加

データ型は必須ではない
↓

**sqlite3によるカラム追加**

```
cur.execute("ALTER TABLE テーブル名 ADD COLUMN カラム名 データ型")
```

**create_table.py**

```
import sqlite3

if __name__ == "__main__":
    con = sqlite3.connect("lec10/databases/pokemon.db")
    cur = con.cursor()
    cur.execute("ALTER TABLE names ADD COLUMN evolvs TEXT")
    con.close()
```

※既存カラムと同じ名前のカラムを追加できない ↓

**実行例**

```
PS C:¥Users¥admin¥Desktop¥ProA2> python ./lec10/create_table.py
Traceback (most recent call last):
    cur.execute("ALTER TABLE names ADD COLUMN evolvs TEXT")
sqlite3.OperationalError: duplicate column name: evolvs
```

【練習】カラムを追加してみよう.

# テーブル名の変更

**sqlite3によるテーブル名変更**

```
cur.execute("ALTER TABLE 旧テーブル名 RENAME TO 新テーブル名")
```

create_table.py

```
import sqlite3

if __name__ == "__main__":
    con = sqlite3.connect("lec10/databases/pokemon.db")
    cur = con.cursor()
    cur.execute("ALTER TABLE names RENAME TO namae")
    con.close()
```

※既存テーブルと同じ名前に変更することはできない ↓

実行例

```
PS C:¥Users¥admin¥Desktop¥ProA2> python ./lec10/create_table.py
Traceback (most recent call last):
    cur.execute("ALTER TABLE names RENAME TO namae")
sqlite3.OperationalError: there is already another table or index with this
name: namae
```

【練習】テーブル名をnamaeに変更してみよう.

# データの挿入（レコードごと）

**sqlite3によるデータ挿入**

```
cur.execute("INSERT INTO テーブル名 VALUES (値1, 値2, 値3, …)")
cur.execute(
    "INSERT INTO テーブル名(カラム名2, カラム名3) VALUES (値2, 値3)"
)

con.commit()
```

← 確定させる

↑
一部のカラムに値を設定する場合
設定しなかったカラムはNoneになる

**insert_data.py**

```
import sqlite3

if __name__ == "__main__":
    con = sqlite3.connect("lec10/databases/pokemon.db")
    cur = con.cursor()
    sql = "INSERT INTO names VALUES (1,'フシギダネ','くさ どく','フシギソウ')"
    cur.execute(sql)
    con.commit()
    con.close()
```

← 確定させないと．．．

※レコードを挿入，更新，削除する際には，
commitメソッドで確定させる必要がある

# データの検索

**sqlite3によるデータ検索と抽出**

```
cur.execute("SELECT カラム名 FROM テーブル名 条件など")
for row in cur:
    print(row)
```

← カーソルをイテレータとして扱う方法

← タプル

```
row  = cur.fetchone()
rows = cur.fetchall()
rows = cur.fetchmany(件数)
```

← fetch系メソッドを用いる方法

**search_data.py**

```
import sqlite3

if __name__ == "__main__":
    con = sqlite3.connect("lec10/databases/pokemon.db")
    cur = con.cursor()
    cur.execute("SELECT  *  FROM names WHERE id > 240 ORDER BY id DESC")
    for row in cur:
        print(row)
    con.close()
```

↑ 全カラムを表す

↑ idが240より大きい行

idがNoneを検索する場合：id IS NULL

↑ 降順：DESC 昇順：ASC

← タプル

# 練習問題：search_data.py

fetchmanyメソッドを使用して，id > 240であるレコードをidの降順に5件抽出し，名前のみ出力するコードを実装せよ．

search_data.py

```
import sqlite3

if __name__ == "__main__":
    con = sqlite3.connect("lec10/databases/pokemon.db")
    cur = con.cursor()
    cur.execute("SELECT * FROM names WHERE id > 240 ORDER BY id DESC")
    for                                                          :

    con.close()
```

実行例

```
PS C:¥Users¥admin¥Desktop¥ProA2> python ./lec10/search_data.py
セレビィ
ホウオウ
ルギア
バンギラス
サナギラス
```

# データの挿入（バルクインサート）

**sqlite3による複数データー括挿入**

```
data = [タプル, タプル]
cur.executemany("INSERT INTO テーブル名 VALUES (?, ?, …)", data)
con.commit()
```

← タプルのリスト　※リストのリストでもOK

**insert_data.py**

```
import sqlite3

if __name__ == "__main__":
    con = sqlite3.connect("lec10/databases/pokemon.db")
    cur = con.cursor()

    data = [(1, 'フシギダネ', 'くさ どく', 'フシギソウ'),
            (2, 'フシギソウ', 'くさ どく', 'フシギバナ'),
            (3, 'フシギバナ', 'くさ どく', ''),]

    sql = "INSERT INTO names VALUES (?, ?, ?, ?)"
    cur.executemany(sql, data)
    con.commit()
    con.close()
```

← タプルのリスト

# 練習問題：insert_data2.py

配布したbase_stats.txtを読み込み，251行あるレコードをテーブルstatsに**バルクインサート**せよ．

ただし，int()により整数型にすること．

insert_data2.py

```
import sqlite3

if __name__ == "__main__":
    con = sqlite3.connect("lec10/databases/pokemon.db")
    cur = con.cursor()
    with open("lec10/data/base_stats.txt", "r") as rfo:
        stats = []
        for row in rfo:
            stat = row.rstrip().split()
            stats.append([int(s) for s in stat])
    
    con.close()
```

# データの削除

**sqlite3によるデータ削除**

```
cur.execute("DELETE FROM テーブル名 条件")
con.commit()
```

**delete_data.py**

```
import sqlite3

if __name__ == "__main__":
    con = sqlite3.connect("lec10/databases/pokemon.db")
    cur = con.cursor()
    cur.execute("DELETE FROM names WHERE id >= 100")
    con.commit()
    con.close()
```

← idが100以上のレコードを削除

# データの更新

**sqlite3によるデータ更新**

```
cur.execute("UPDATE テーブル名 SET カラム名1=値1, カラム名2=値2 条件")
cur.execute("UPDATE テーブル名 SET カラム名1=?, カラム名2=?", タプル)
con.commit()
```

↑
リストでもOK

**update_data.py**

```
import sqlite3

if __name__ == "__main__":
    con = sqlite3.connect("lec10/databases/pokemon.db")
    cur = con.cursor()
    cur.execute("UPDATE names SET name='フシミダネ' WHERE id == 1")
    cur.execute("UPDATE names SET name=? WHERE id == 2", ("フシミソウ", ))
    con.commit()
    con.close()
```

↑
idが1のレコードのnameを'フシミダネ'に更新
idが2のレコードのnameを'フシミソウ'に更新

↑
※カンマ「,」がないとタプルだと認識されない
↓

**実行例**

```
PS C:¥Users¥admin¥Desktop¥ProA2> python ./lec10/update_data.py
Traceback (most recent call last):
    cur.execute("UPDATE names SET name=? WHERE id == 2", ("フシミソウ"))
sqlite3.ProgrammingError: Incorrect number of bindings supplied.
The current statement uses 1, and there are 5 supplied.
```

# テーブルの確認

**sqlite3によるテーブル確認**

```
cur.execute("SELECT * FROM sqlite_master WHERE type='table'")
for row in cur:
    print(row)
```

**show_table.py**

```
import sqlite3

if __name__ == "__main__":
    con = sqlite3.connect("lec10/databases/pokemon.db")
    cur = con.cursor()
    cur.execute("SELECT * FROM sqlite_master WHERE type='table'")
    for row in cur:
        print(row)
    con.close()
```

**実行例**

```
PS C:¥Users¥admin¥Desktop¥ProA2> python ./lec10/show_table.py
('table', 'names', 'names', 3, 'CREATE TABLE names (id INTEGER, name TEXT,
                                                    types TEXT, evolvs TEXT)')
('table', 'stats', 'stats', 7, 'CREATE TABLE stats (hp, atk,
                                                   dfs, sp_atk, sp_dfs, spd)')
```

# その他：Rowオブジェクト

**sqlite3.Rowによる結果オブジェクト作成**

```
con.row_factory = sqlite3.Row

cur.execute("SELECT カラム FROM テーブル名 条件など")
for row in cur:
    print(row["カラム名"])
```

← Connectionオブジェクトのrow_factory属性にRowクラスオブジェクトを設定

**search_data3.py**

```
import sqlite3

if __name__ == "__main__":
    con = sqlite3.connect("lec10/databases/pokemon.db")
    con.row_factory = sqlite3.Row
    cur = con.cursor()
    cur.execute("SELECT * FROM names")

    for row in cur.fetchmany(5):
        print(row["name"])
    con.close()
```

**実行例**

```
python ./lec10/search_data3.py
フシミダネ
フシミソウ
フシギバナ
ヒトカゲ
リザード
```

# ユーザ定義関数

**sqlite3による関数作成**

```
def ユーザ定義関数(引数の数分の引数):
    return 値
```

↓ Connectionオブジェクトに関数を登録する

```
con.create_function(SQL文内での関数名, 引数の数, ユーザ定義関数)

cur.execute("SELECT 関数名(引数の数分のカラム) FROM テーブル名")
```

**user_function.py**

```
import sqlite3
def sum_stats(h, a, b, c, d, s):
    return h+a+b+c+d+s

if __name__ == "__main__":
    con = sqlite3.connect("lec10/databases/pokemon.db")
    con.create_function("sum", 6, sum_stats)
    cur = con.cursor()
    cur.execute("SELECT sum(hp, atk, dfs, sp_atk, sp_dfs, spd) FROM stats")
    for row in cur.fetchall():
        print(row)
    con.close()
```

**実行例**

```
python ./lec10/user_function.py
(318,)
(405,)
(525,)
 :
```

# 練習問題：user_function2.py

こうげきatkとぼうぎょdfsの平均値が高い順にレコードを表示させよ.

1. 2つの値を受け取り，その平均値を返す関数balanceを定義する
2. Connectionクラスのインスタンスにbalance関数をavgという名前で登録する
3. avg関数にatkとdfsを渡し，その値に基づいて降順に出力するSELECT文を実行する

   ※SQL文の「avg() AS 名前」と「ORDER BY 名前 DESC」を組み合わせる
4. for-in文で10件のみ出力する

# 解答例：user_function2.py

user_function2.py

```
import sqlite3

def balance(x, y):
    return (x+y)/2

if __name__ == "__main__":
    con = sqlite3.connect("lec10/databases/pokemon.db")
    con.
    cur = con.cursor()
    cur.execute("SELECT
                AS avg FROM stats ORDER BY avg DESC")
    for row in
        print(row)
    con.close()
```

conにbalance関数をavgという名前で登録する

avg関数にatkとdfsを渡す

10件表示

実行例

```
python ./lec10/user_function2.py
(142.5,)
(137.5,)
(125.0,)
(122.5,)
(122.0,)
(120.0,)
:
```

# ユーザ定義集計関数

複数のレコードに対する集計

**sqlite3による集計関数作成**

```
class 集計クラス:
    def step(self, 引数の数分の引数):
        各レコードに対する処理

    def finalize(self):
        return 対象の全レコードに関する集計結果
```

↓ Connectionオブジェクトに集計関数（実態はクラス）を登録する

```
con.create_aggregate(関数名, 引数の数, 集計クラス)

cur.execute("SELECT 関数名(引数の数分のカラム) FROM テーブル名")
```

# コード例：user_aggregate.py

user_aggregate.py

```
class MaxAverage:
    def __init__(self):
        self.max = 0

    def step(self, *tpl):
        avg = sum(tpl)/len(tpl)
        if self.max < avg:
            self.max = avg

    def finalize(self):
        return self.max

if __name__ == "__main__":
    con = sqlite3.connect("lec10/databases/pokemon.db")
    con.create_aggregate("max_avg", 6, MaxAverage)
    cur = con.cursor()
    cur.execute("SELECT max_avg(hp, atk, dfs, sp_atk, sp_dfs, spd) FROM stats")
    for row in cur:
        print(row)
    con.close()
```

← 各レコード（今回は種族値）の平均値を求め
← 現時点の最大値をself.maxに保持

← 対象全レコードの集計結果（今回は平均値の最大値）

↑
対象となる全251レコードに対して
1レコードずつstepメソッドを実行

全レコードに対する処理が終わったら
finalizeメソッドが集計結果をreturn