DB スクリプティング（Python、CGI）

目次

DB スクリプティング（Python、CGI）

はじめに

スクリプトには脚本という意味があり、コンピュータの世界では仕事の手順と簡単な例外処理の扱いを記述したものをスクリプトと呼びます。Unix/Linux ではシェルスクリプトで grep や sed、awk 等の単機能プログラムを部品として使い複雑な処理を実現します。Windows では同様の目的の bat ファイルや VBScrpt、PowerShell があります。これらのスクリプト言語が扱えるデータは主に一般ファイルかそれに類するものですが、Web 開発に利用されるようになった Perl、PHP、Ruby、Javascript（Node.js）、Python 等は RDB や NoSQL データベースが扱えるようになっています。

後者はインタープリタ言語やプログラミング言語と呼んだ方が適当かもしれませんが、型宣言なしに変数が使えてソースのまま実行できるというスクリプトの手軽さを活かしてデータベースを使う例を紹介します。

※ いくつかの言語は、型を指定できるように拡張したり JIT コンパラが存在します

1. Python で使える DB

オープンソースの PostgreSQL や MySQL 他、商用の Oracle や SQLServer 用のライブラリが作られていてるので、それぞれインストールして使うことができます。

標準ライブラリ（以下、Python 3.10.6 が前提）には SQLite を扱うことができる sqlite3 が用意されていて、import するだけで使うことができます。

SQLite はローカルファイルをデータベースにするライブラリで、リモートサーバは存在しません。281 テラバイトまで扱えて 1 テラバイト未満での使用を推奨しています[1]。オープンソース／パブリックドメインで改変や商用利用を無制限に認めている[2]ため、公式サイトによれば各種のブラウザやスマートフォン（Android、iPhone）、その他の PP 製品に組み込まれており、ユーザの権限管理や View への書き込み等の機能を省力している以外は標準的な SQL 言語を使うことができます。

以下に標準ライブラリ sqlite3 を使って DB にアクセスする例を紹介します。SQLite は DB Browser for SQLite という専用ツールや DBeaver 等の汎用的なデータベースツールで扱えます。

2. CSV から DB へ追加

ファイルダイアログを使って CSV ファイルを幾つか選択し、データベースに登録します。

データベースが作られていない場合は、データベース（ローカルファイル）を作りテーブルを作成します。

2.1. CSV とテーブルの項目の並びが一致している例

項目の並びが一致している CSV ファイルを集めて一つのテーブルに集約します。

※CSV の内容は以下のような見出し行付きの内容を想定しています（見出し行は読み飛ばします）

名前,１月,２月,３月,４月,５月,６月
太郎,100,200,300,400,500,600
次郎,110,220,330,440,550,660

---

[1] Appropriate Uses For SQLite（SQLite の適切な使用） https://www.sqlite.org/whentouse.html

[2] SQLite Is Public Domain https://www.sqlite.org/copyright.html

csvdb_simple.py

```
"""
テーブル／データ登録
選択画面から１つまたは複数の CSV ファイルを選び、テーブルに登録します。
・データベースは sqlite3 を使い、ファイルが無ければ固定の場所に作ります
【前提条件】
・CSV ファイルの項目はテーブルのカラム定義と順が一致し抜けが無いこと

"""
import tkinter
import tkinter.filedialog # filedialog はサブモジュールなので個別に import 要
import csv
import sqlite3
from pathlib import Path
import logging

logging.basicConfig(level=logging.INFO) # CRITICAL,ERROR,WARNING,INFO,DEBUG

csvfs = tkinter.filedialog.askopenfilenames(title='入力ファイル選択'
                                            , filetypes=[('csv', '*.csv')]
                                            )

dbfile = 'C:/Users/User/Desktop/Python-Codes/dbdata.db'
# db が無かったら db とテーブルを作る
dbobj = Path(dbfile)
if not dbobj.is_file():
    try:
        dbobj.touch()
        con = sqlite3.connect(dbfile)
        # Python 公式によると、excute 等には connect オブジェクトを使った方が
        # cursor を使うよりも効率が良いそうです
        con.execute('''CREATE TABLE collect_tbl(
                        名前 VARCHAR(15), １月 NUMERIC, ２月 NUMERIC, ３月 NUMERIC
                      , ４月 NUMERIC, ５月 NUMERIC, ６月 NUMERIC)
                    '''
                   )
    except Exception as e:
        print('Exception !! ', e)
        con.close()
        dbobj.unlink()
    else:
        print('created db>',dbfile)
else:
    con = sqlite3.connect(dbfile)

''' 小数点"."とカンマ区切りは数値として許容する '''
def is_num_dotcomma(s):
    try:
        float(s.replace(',', ''))
    except ValueError:
        return False
    else:
        return True
```



```
53
54  outcnt = incnt = 0
55  for csvf in csvfs:
56      with open(csvf, 'r', encoding='utf-8') as f:
57          csvr = csv.reader(f)
58          for r in csvr:
59              incnt += 1
60              if is_num_dotcomma(r[1]): # 見出し行を除外するため 2 列目が数字かチェック
61                  # table に各行のデータを挿入
62                  con.execute('INSERT INTO collect_tbl VALUES (?,?,?,?,?,?,?);', r)
63                  logging.debug('insert %s', r[0])
64                  outcnt += 1
65              else:
66                  logging.info('ommit %s,%s', r[0], r[1])
67  con.commit()
68  con.close()
69  print(f'処理件数 in={incnt}, out={outcnt}')
```

＜使っているライブラリ＞

9~10 行 tkinter とそのサブモジュールが GUI とファイル選択ダイアログのライブラリ

11 行 csv ファイルを扱うライブラリ

12 行 sqlite3 RDB の DB-API を提供するライブラリ

13 行 pathlib の Path ファイルパスのオブジェクトを作るライブラリ

＜主な処理＞

18~20 行 ファイル選択ダイアログを表示し、選択したファイルがリストで返ってきます

22 行 データベースが作られるファイルパス

25~41 行 22 行で指定したファイルが存在しないとき、27 行の touch()でファイルを作り sqlite3 のコネクションを使って create table を実行します

SQLite は列のタイプ（型）を指定しなくてもテーブルが作れますが推奨型を指定することもでき、NUMERIC で定義するとデータ格納時に整数なら INT、小数点があると FLOAT、数値で扱えない場合は TEXT として格納されます

36 行 上記の一連の処理中に例外が発生したら、コネクションを閉じて touch()で作ったファイルを unlink()で消します

43 行 既にデータベースのファイルが存在する場合は、コネクションを作ります

46 行 数字チェック用の関数を定義しています。数字か否かを確認する以下の関数がありますが、

オブジェクト.isdecimal() .isdigit() . isnumeric()

これらは文字列に小数点を含んでいると False が返るので、型変換で例外が発生するか否かを数値判定に使っています

62 行 INSERT 文の VALUES()をプレースホルダー"?"で指定します。レコード"r"の中の項目はテーブル定義と並びを一致させる必要があります

2.2. CSV の見出し行を使ってテーブルに登録する例

テーブルの項目名と CSV の見出し行に書いた項目名を関係づけて登録します。

csvdb_freestyle.py

```
"""
テーブル／データ登録
選択画面から１つまたは複数の CSV ファイルを選び、テーブルに登録します。
・データベースは sqlite3 を使い、ファイルが無ければ固定の場所に作ります
【前提条件】
・CSV ファイルの UTF-8(BOM 付き／BOM なし)が open 文の encoding と一致していること
・CSV ファイルにテーブルのカラム名と一致させた見出し行が付いていること
（名前が一致した項目だけを登録し、ない項目は null になります）

"""
import tkinter
import tkinter.filedialog # filedialog はサブモジュールなので個別に import 要
import csv
import sqlite3
from pathlib import Path
import logging

logging.basicConfig(level=logging.DEBUG) # CRITICAL,ERROR,WARNING,INFO,DEBUG

csvfs = tkinter.filedialog.askopenfilenames(title='入力ファイル選択'
                                            , filetypes=[('csv', '*.csv')]
                                            )

dbfile = 'C:/Users/User/Desktop/Python-Codes/dbdata.db'
# db が無かったら db とテーブルを作る
dbobj = Path(dbfile)
if not dbobj.is_file():
    try:
        dbobj.touch()
        con = sqlite3.connect(dbfile)
        # Python 公式によると、excute 等には connect オブジェクトを使った方が
        # cursor を使うよりも効率が良いそうです
        con.execute('''CREATE TABLE collect_tbl(
                        名前, １月, ２月, ３月, ４月, ５月, ６月
                        )
                    '''
                    )
    except Exception as e:
        print('Exception !! ', e)
        con.close()
        dbobj.unlink()
    else:
        print('created db>',dbfile)
else:
    con = sqlite3.connect(dbfile)

# csv の見出し行／項目名と関係付けるために、テーブルのカラム名を取り出す
cur = con.execute('SELECT * FROM collect_tbl limit 1')
```



```
49  fields = [row[0] for row in cur.description]
50  query = 'INSERT INTO collect_tbl(' + ','.join(fields) ¥
51          + ') VALUES(' + ':' + ',:'.join(fields) + ');' # 文字列は¥で継続行にできます
52
53  logging.debug(query)
54
55  # 選択された csv ファイルを読んで全件をテーブルに insert
56  outcnt = incnt = 0
57  for csvf in csvfs:
58      # Excel から出力した csv は BOM 付きになるので encoding に'utf-8_sig'を指定します。
59      # メモ帳で開いて右下に(BOM 付き)とでなければ、'utf-8'を指定します
60      with open(csvf, 'r', encoding='utf-8_sig') as f:
61          csvr = csv.DictReader(f)
62          for r in csvr:
63              logging.debug(r)
64              incnt += 1
65              cur.execute(query, r)
66              outcnt += 1
67
68  con.commit()
69  con.close()
70  print(f'処理件数 in={incnt}, out={outcnt}')
```

「2.1 CSV とテーブルの項目の並びが一致している例」との違いは、主に以下の部分です。

60 行 ファイルを開くときの encoding を BOM 付きの"utf-8_sig"に変更

…（BOM 付き）はファイルの先頭に¥ufeff が付加されますが、"utf-8"では Python がファイルの一部として読み込んでしまします

```
with open(csvf, 'r', encoding='utf-8') as f:
    csvr = csv.DictReader(f)
    for r in csvr:
        print(r)
```

```
{'¥ufeff名前': '太郎', '1 月': '100', '2 月': '200', '3 月': '300', '4 月': '400', '5 月': '500', '6 月': '600'}
```

61 行 csv ファイルの読み込みに DictReader()を使い、項目見出しをキーにした dict 型で読み込む

65 行 qurey（insert 文)を以下のようにします

```
'INSERT INTO collect_tbl(名前,１月,２月,３月,４月,５月,６月) VALUES(:名前,:１月,:２月,:３月,:４月,:５月,:６月);'
```

VALUES()の中は 61 行で読み込んだ dict 型のキーの先頭に":"を付けて、"名前付パラメータ"をプレースホルダーとして使っています…テーブルの列名と dict のキーで一致する項目があれば dict の値が登録され、一致しない項目には null が登録されます

※ 名前付パラメータには、ここで使っている":"以外にも"@"と"$"が使えます･･･@名前,$名前 等

DB スクリプティング（Python、CGI）

3. DB の参照

データベースに登録した内容はデータベースツールでも確認できますが、スクリプトでも簡単に取り出してデータを加工することができます。

3.1. CGI を使った参照

CGI（Common Gateway Interface）は Web サーバから起動されて処理結果（html）を Web サーバを経由してクライアントに返す方式です。Python を使うと以下のようになります。

dbdisp.py（cgi で起動するモジュール）

```
1   #!/usr/bin/env python
2
3   import cgitb
4   cgitb.enable()
5
6   import logging
7   logging.basicConfig(level=logging.DEBUG) # CRITICAL,ERROR,WARNING,INFO,DEBUG
8
9   import sqlite3
10  dbfile = 'C:/Users/User/Desktop/Python-Codes/dbdata.db'
11  sql = 'SELECT * FROM collect_tbl'
12
13  print("""Content-type: text/html¥n
14  <!DOCTYPE html>
15  <html>
16  <body>""")
17
18  # データベースに接続し、グローバル変数で定義した sql を実行します
19  con = sqlite3.connect(dbfile)
20  cur = con.cursor()
21  cur.execute(sql)
22  # 検索結果から列名（discription）を取り出して html の表にします
23  col_names = [cn[0] for cn in cur.description]
24  print('<table border=`1``>')
25  print('<tr>')
26  for cn in col_names:
27      print('<th>', cn, '</th>')
28
29  print('</tr>')
30  # 検索結果から列のデータを取り出して html の表にします
31  rows = cur.fetchall()
32  for row in rows:
33      print('<tr>')
34      for col in row:
35          print('<td>', col, '</td>')
36
37  print('</tr>')
38  print('</table>')
39  print ("""</body>
40  </html>""")
41  con.close()
```


DB スクリプティング（Python、CGI）

CGI を使ってモジュールを呼び出すサーバは、以下となります。

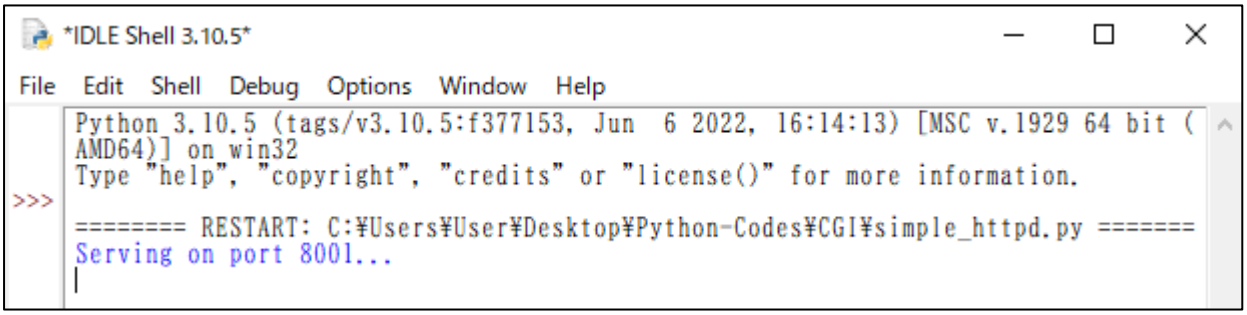
simple_httpd.py（cgi を制御するサーバ）

```
1  from http.server import CGIHTTPRequestHandler, HTTPServer
2
3  class Handler(CGIHTTPRequestHandler):
4      cgi_directories = ["/py-cgi"]
5
6  PORT = 8001
7  httpd = HTTPServer(("", PORT), Handler)
8  print(f"Serving on port {PORT}...")
9  httpd.serve_forever()
```

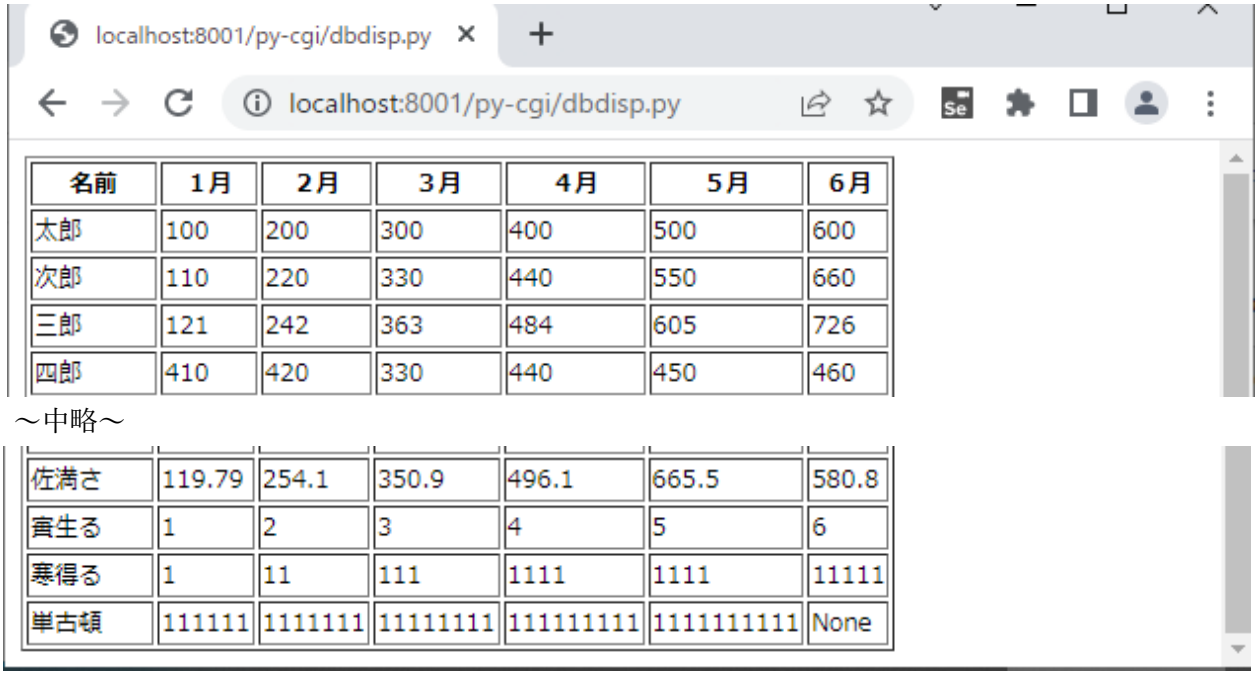
サーバとモジュールは以下の構成にします。

```
～/サーバフォルダ
 |  simple_httpd.py      …サーバ
 |
 └─py-cgi                …フォルダ名はサーバの cgi_directories と一致していれば OK
       dbdisp.py         …html を生成するモジュール
```

サーバを起動し、正常に起動するとメッセージがでます（以下は IDLE Editor から起動した場合）。



http://localhost:8001/py-cgi/dbdisp.py にブラウザからアクセスすると…



名前	1月	2月	3月	4月	5月	6月
太郎	100	200	300	400	500	600
次郎	110	220	330	440	550	660
三郎	121	242	363	484	605	726
四郎	410	420	330	440	450	460

～中略～

佐満さ	119.79	254.1	350.9	496.1	665.5	580.8
賓生る	1	2	3	4	5	6
寒得る	1	11	111	1111	1111	11111
単古頓	111111	1111111	11111111	111111111	1111111111	None

3.2. CGI を使った参照（ページング）

データが大量で、リクエスト／レスポンスを小分けにしてページングの操作を加えた例です。

dbdisp_page.py（ページングする cgi モジュール）

```
1   #!/usr/bin/env python
2
3   import os
4   import cgitb
5   cgitb.enable()
6
7   import logging
8   logging.basicConfig(level=logging.DEBUG) # CRITICAL,ERROR,WARNING,INFO,DEBUG,
9
10  import sqlite3
11  dbfile = 'C:/Users/User/Desktop/Python-Codes/dbdata.db'
12  limit = 10 # 出力件数／ページ
13  sql = 'SELECT row_number() OVER () r_no, * FROM collect_tbl limit ' + str(limit)
14
15  def str2int(s):
16      try:
17          return int(s)
18      except ValueError:
19          return 0
20
21  def cre_head():
22      print("""Content-type: text/html¥n
23      <!DOCTYPE html>
24      <html>
25      <body>""")
26
27
28  def display_data(move='', rec='1', **_): # **_で余計なパラメータがあったら拾う
29      cPos = str2int(rec)
30      match move:
31          case "fowd":
32              sPos = cPos + limit
33
34          case "prev":
35              sPos = cPos - limit
36              if sPos < 1 :sPos = 1
37
38          case _:
39              sPos = cPos
40
41      ex_sql = sql + ' offset ' + str(sPos - 1)
42      logging.debug('sql=%s', ex_sql)
43
44      con = sqlite3.connect(dbfile)
45      cur = con.cursor()
46      cur.execute(ex_sql)
47      col_names = [cn[0] for cn in cur.description]
48      print('<table border=`1``>')
```



```
49      print('<tr>')
50      for cn in col_names:
51          print('<th>', cn, '</th>')
52
53      print('</tr>')
54
55      rows = cur.fetchall()
56      for row in rows:
57          print('<tr>')
58          for col in row:
59              print('<td>', col, '</td>')
60
61      print('</tr>')
62      print('</table>')
63
64      con.close()
65
66      return cPos if len(rows) == 0 else sPos
67
68  def cre_tail(sPos):
69      print('<form method="get" action="dbdisp_page.py" target="_top">')
70      print('<button type=submit name="move" value="go" > GoTo </button>')
71      print('<input type="text" name="rec" value="' + str(sPos) + '">')
72      print('<button type=submit name="move" value="prev" > prev </button>')
73      print('<button type=submit name="move" value="fowd" > fowd </button>')
74      print('</form>')
75
76      print ("""</body>
77      </html>""")
78
79  def main():
80      que_para = os.environ.get('QUERY_STRING')
81      logging.debug('QUERY_STRING %s', que_para)
82      if que_para == None or que_para.find('=') == -1:
83          logging.debug('** Request parameter nothing ** goto first !! ')
84          form = {"move" : "first", "rec" : "1"}
85      else:
86          # url パラメータは、key1=val1&key2=val2 の形式で QUERY_STRING に入っているので
87          # "&"でパラメータ分割⇒"="で key/val 分割を行い、KEY は小文字にして{key1:val1, key2:val2}
88          paras = que_para.split('&')
89          form = {key.lower() : val for key, val in (para.split('=') for para in paras)}
90          logging.debug('form %s', form)
91
92      cre_head()
93      sPos = display_data(**form) # **で辞書(form)をキーワードパラメータに変換
94      cre_tail(sPos)
95
96  main()
```

追加した主な機能は、以下の点です。

・SELECT 文に limit 句と offset 句を加えて抽出の範囲を制限する

・出力する html にレコード番号表示と番号指定入力、遷移ボタンを追加する（cre_tail 関数）

DB スクリプティング（Python、CGI）

サーバとモジュールは以下の構成にします。

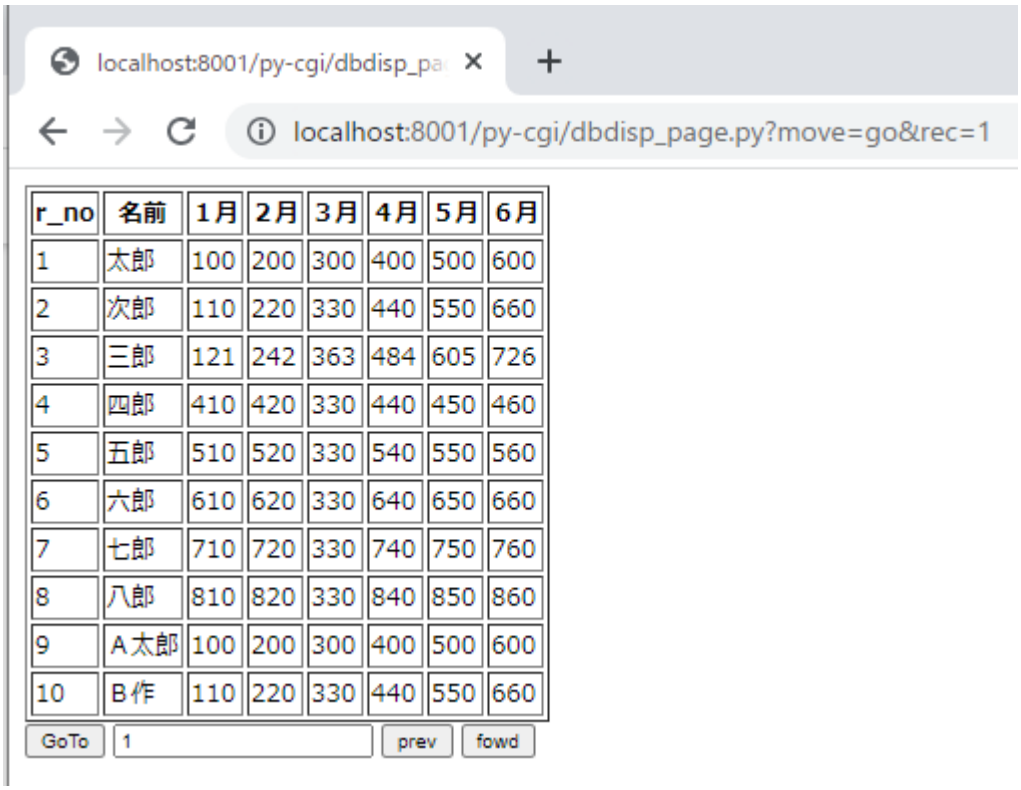
```
～/サーバフォルダ
 |   index.html          …追加した html
 |   simple_httpd.py
 |
 └──py-cgi
        dbdisp_page.py  …追加したモジュール
```

※index.html はデフォルトで表示されるページで、ここに dbdisp_page.py へのリダイレクトと初回表示用のパラメータを書いておきます。内容は、以下のようにします。

【index.heml】

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
<!--
location.href="py-cgi/dbdisp_page.py?move=go&rec=1";
-->
</script>
</html>
```

http://localhost:8001/ にブラウザからアクセスすると、リダイレクトして以下のようになります。



r_no	名前	1月	2月	3月	4月	5月	6月
1	太郎	100	200	300	400	500	600
2	次郎	110	220	330	440	550	660
3	三郎	121	242	363	484	605	726
4	四郎	410	420	330	440	450	460
5	五郎	510	520	330	540	550	560
6	六郎	610	620	330	640	650	660
7	七郎	710	720	330	740	750	760
8	八郎	810	820	330	840	850	860
9	A太郎	100	200	300	400	500	600
10	B作	110	220	330	440	550	660

※実際はパラメータが指定されない場合は 1 件目から表示するようにモジュールを作ってあるので、直接 http://localhost:8001/py-cgi/dbdisp_page.py にアクセスしても上記の画面が表示されます

3.3. SQL、CGI や Web に関わる Python とその他の一般的な注意点

CGI は Web サーバとは別のプロセスが起動して http リクエストを処理します。

プロセスの起動は OS が行い環境変数を使ってリクエストパラメータを渡すのですが、このリクエストの受け取りに関する Python の cgi という標準ライブラリが将来のバージョンでの廃止候補になっています（このため、記載したコードでは使っていません）。

CGI の OS が関与し環境変数が使われることを利用した OS コマンド インジェクションの脆弱性が 2014 年に見つかっています。これは GNU Bash(Ver 4.3 以前)で環境変数に OS コマンドを設定する（CGI のリクエストパラメータにコマンド文字列を入れておく）とコマンドとして実行してしまうというものです。この件に関しては Bash のバージョンアップで回避できます。

CGI に限らないその他のサーバ攻撃方法として代表的なのが、html インジェクションと SQL インジェクションです。html インジェクションは html のタグと JavaScript を入力項目にセットし、次の確認画面の描画時に攻撃が動作するというもので、これから開発するシステムに関しては最新の有名どころのフレームワークを使えば回避（サニタイジング）できます。

SQL インジェクションは画面やファイルから入力した内容をそのまま SQL（クエリ）の文字列に埋め込んでしまうと発生します。例えば、ユーザ ID や商品コード等の RDB に問合せを発行しそうな項目に SQL 文の終端文字とそれに続けて DELETE や DROP という文字列を入力されると環境を破壊される恐れがあります。これは SQL を一旦解析（Java の場合は PreparedStatement）し、プレースホルダーを使って可変の値を合成することで攻撃を回避できます。PreparedStatement は主要な言語と RDBMS は同様の機能を持っていて、今回のコード例のように Python では標準でプレースホルダーを使うことができます。

いずれも ユーザが入力した文字列を生で OS やブラウザ、DBMS に評価させてしまうことで危険になるのですが、フレームワークでサニタイジングを行っても行わなくてもプレースホルダーを使っても使わなくても“正しい”操作に対しては同じ動作をするので専用の試験が必要です。システムの特性により必要になる対策は異なりますが、恐らく一番安価で効果がある対策は新人教育です。

以上

