# プログラミングA2
# 第12回

**練習問題解答例**

# 練習問題：html.py

HTMLの<ul>…</ul>タグと<li>…</li>タグで文字列を囲むデコレータを実装せよ.

[要件]

1. get_title関数
   - Monsterクラスのインスタンスを受け取り,
   - インスタンスからtitle属性を取得し,
   - そのtitle文字列を返す
2. concat_strs関数　　stringsをconcatenateするの意味
   - Monsterクラスのインスタンスがならぶリストを受け取り,
   - 各要素に対してget_title関数を用いて文字列を取得し,
   - それらを結合した文字列を返す
3. li_decorator関数
   - 受け取った関数を実行する前後に<li>, </li>を付与するデコレータ
4. ul_decorator関数
   - 受け取った関数を実行する前後に<ul>, </ul>を付与するデコレータ関数
5. 3のデコレータで1の関数をデコレートする
6. 4のデコレータで2の関数をデコレートする

# 解答例：html.py

デコレータ
デコレート対象
デコレート中
デコレート後

html.py：デコレート対象の関数

```
@li_decorator
def get_title(mon):
    return mon.title


@ul_decorator
def concat_strs(mon_lst):
    s = []
    for mon in mon_lst:
        s.append(get_title(mon))
    return "".join(s)
```

html.py：デコレータ関数

```
def li_decorator(func):
    def _func(t):
        s1 = "<li>"
        s2 = func(t)
        s3 = "</li>"
        return s1+s2+s3+"¥n"
    return _func


def ul_decorator(func):
    def _func(t):
        s1 = "<ul>"
        s2 = func(t)
        s3 = "</ul>"
        return s1+s2+s3+"¥n"
    return _func
```

# 実行例：html.py

html.py

```
if __name__ == "__main__":
    monsters = [
        Monster("イーブイ"),
        Monster("シャワーズ"),
        Monster("サンダース"),
        Monster("ブースター"),
        Monster("エーフィ"),
        Monster("ブラッキー"),
        Monster("リーフィア"),
        Monster("グレイシア"),
        Monster("ニンフィア"),
    ]


print(concat_strs(monsters))
```

実行例

```
<ul><li>イーブイ</li>
<li>シャワーズ</li>
<li>サンダース</li>
<li>ブースター</li>
<li>エーフィ</li>
<li>ブラッキー</li>
<li>リーフィア</li>
<li>グレイシア</li>
<li>ニンフィア</li>
</ul>
```

デコレートしなかった場合も確認してみよう.

# 練習問題：time_measure.py

関数を実行する前後の時刻を測り，差分を求めることで
関数の所要時間を表示するデコレータを実装せよ

time_measure.py

```
@time_measure
def str_concat1(num):
    s = ""
    for i in range(num):
        s += str(i)

@time_measure
def str_concat2(num):
    s = "".join([str(i) for i in range(num)])

if __name__ == "__main__":
    str_concat1(1000000)
    str_concat2(1000000)
```

← 本練習問題で実装するデコレータによりデコレートしている

空文字列sにnum個の数字を1つずつ連結：
**新たなstrオブジェクトがnum回生成されるので遅い**

← デコレートしないとどうなる？

**実行例**

```
所要時間：1.09秒
####################
所要時間：0.16秒
####################
```

# 解答例：time_measure.py

**time_measure.py：デコレータ部分**

```
def time_measure(func):
    def _func(*args):
        bgn = time.time()
        func(*args)
        end = time.time()
        print(f"所要時間：{end-bgn:.2f}秒")
        print("#"*20)
    return _func
```

デコレート対象関数の引数

デコレータは，
デコレート対象の関数funcを引数に取り
デコレータ内で
デコレートした（＝少し改良した）新たな関数を作り返す

# 練習問題：table_creator.py

sqliteのデータベースにテーブルを構築するクラスを
コンテキストマネージャとして実装せよ

table_creator.py

```
class TableCreator:
    def __init__(self, db_path):
        self.db_path = db_path    ← データベース名

    def __enter__(self):
        self.con = sqlite3.connect(self.db_path)
        print(f"{self.db_path}に接続しました")
        self.cur = self.con.cursor()
        return self

    def __exit__(self, exc_type, exc_value, traceback):
        if exc_type is None:
            print("正常にSQLが実行されました")
            self.con.close()
            print("切断しました")
        else:
            print(f"正常にSQLが実行されませんでした：{exc_type} {exc_value}")
            self.con.close()
            print("切断しました")
```

# 実行例：table_creator.py

table_creator.py

```
    def create(self, tbl_name, col_lst):
        self.tbl_name = tbl_name
        self.columns = ",".join(col_lst)
        sql = f"CREATE TABLE {self.tbl_name} ({self.columns})"
        self.cur.execute(sql)
        print(f"{self.tbl_name}を構築しました")

if __name__ == "__main__":
    db_path = "lec12/data/pokemon.db"
    tbl_name = "names"
    col_lst = ["id", "name", "types", "evolvs"]
    with TableCreator(db_path) as tc:
        tc.create(tbl_name, col_lst)
```

← テーブルを構築するメソッド

← テーブル名
← カラムリスト

← テーブルを構築する

- 正常にテーブルを構築できた場合

実行例

```
lec12/data/pokemon.dbに接続しました
namesを構築しました
正常にSQLが実行されました
切断しました
```

- 正常にテーブルを構築できなかった場合でもDBからの切断は必ず実行される

実行例

```
lec12/data/pokemon.dbに接続しました
正常にSQLが実行されませんでした：<class
'sqlite3.OperationalError'> table names
already exists
切断しました
```