【新人エンジニア向け】Pythonで特定の関数だけ実行する方法

はじめに

コードを書いていると、「この関数だけテストしたい」「他の関数は動かさずに、この部分だけ確認したい」という場面がよくあります。この記事では、Pythonで特定の関数だけを実行する実践的な方法を紹介します。

方法1: 関数を直接呼び出す（最もシンプル）

最も基本的な方法は、実行したい関数だけを呼び出すことです。

```python
def calculate_tax(price):
    """消費税を計算する関数"""
    return price * 0.1

def calculate_total(price):
    """合計金額を計算する関数"""
    tax = calculate_tax(price)
    return price + tax

def send_email(total):
    """メールを送信する関数（テスト中は実行したくない）"""
    print(f"メール送信: 合計{total}円")

# テストしたい関数だけ呼び出す
result = calculate_tax(1000)
print(f"消費税: {result}円")

# この関数は呼び出さなければ実行されない
# send_email(11000)  # コメントアウト
```

メリット: シンプルで分かりやすい

デメリット: 実行の度にコードを編集する必要がある

方法2: if name == "main": を活用する

Pythonの定番パターンです。このブロック内のコードは、スクリプトを直接実行したときだけ動きます。

```python
```

```python
def data_processing():
    """データ処理"""
    print("データを処理中...")
    return [1, 2, 3, 4, 5]

def save_to_database(data):
    """データベースに保存"""
    print(f"DBに保存: {data}")

def send_notification():
    """通知を送信"""
    print("通知を送信しました")

if __name__ == "__main__":
    # ここでテストしたい関数だけ実行
    data = data_processing()
    print(f"処理結果: {data}")

    # テスト中は実行したくない関数はコメントアウト
    # save_to_database(data)
    # send_notification()
```

使い分けのポイント:

- モジュールとして他のファイルからimportされる可能性がある場合に有効
- `python script.py`で実行したときだけ動く

方法3: 関数名を指定して実行（少し応用）

コマンドライン引数で実行する関数を選べるようにします。

```python
```

```python
import sys

def test_function_a():
    print("関数Aを実行")
    # テストコード

def test_function_b():
    print("関数Bを実行")
    # テストコード

def test_function_c():
    print("関数Cを実行")
    # テストコード

if __name__ == "__main__":
    # 関数名の辞書を作成
    functions = {
        'a': test_function_a,
        'b': test_function_b,
        'c': test_function_c
    }

    if len(sys.argv) > 1:
        func_key = sys.argv[1]
        if func_key in functions:
            functions[func_key]()
        else:
            print(f"エラー: '{func_key}'は存在しません")
            print(f"利用可能な関数: {', '.join(functions.keys())}")
    else:
        print("使い方: python script.py [a|b|c]")
```

実行方法:

```bash
python script.py a  # 関数Aだけ実行
python script.py b  # 関数Bだけ実行
```

メリット: コードを編集せずに実行する関数を切り替えられる

方法4: フラグを使った条件分岐

設定ファイルのようにフラグで制御する方法です。

```python
```

```python
# 実行制御フラグ
RUN_DATA_FETCH = True
RUN_DATA_PROCESS = True
RUN_SEND_EMAIL = False  # これだけ無効化

def fetch_data():
    print("データ取得中...")
    return {"name": "商品A", "price": 1000}

def process_data(data):
    print(f"データ処理中: {data}")
    return data

def send_email(data):
    print(f"メール送信: {data}")

if __name__ == "__main__":
    data = None

    if RUN_DATA_FETCH:
        data = fetch_data()

    if RUN_DATA_PROCESS and data:
        data = process_data(data)

    if RUN_SEND_EMAIL and data:
        send_email(data)
```

メリット: 複数の関数の実行を柔軟に制御できる

実践的なTips

1. デバッグプリントを活用する

```python
def my_function():
    print(">>> my_function 開始")  # 開始を明示
    result = 100 * 2
    print(f">>> 計算結果: {result}")  # 途中経過を表示
    return result

my_function()
```

2. try-exceptで安全に実行

```python
def risky_function():
    # エラーが起きるかもしれない処理
    return 10 / 0

if __name__ == "__main__":
    try:
        result = risky_function()
        print(f"成功: {result}")
    except Exception as e:
        print(f"エラー発生: {e}")
        # エラーが起きても他の関数は実行できる
```

3. テスト用の小さな入力データを用意

```python
def process_large_data(data_list):
    """大量データを処理する関数"""
    for item in data_list:
        # 重い処理
        pass

if __name__ == "__main__":
    # 本番は10000件だが、テストは3件だけ
    test_data = [1, 2, 3]
    process_large_data(test_data)
```

まとめ

関数を個別に実行する方法はいくつかありますが、状況に応じて使い分けましょう。

- **簡単なテスト**: 方法1（直接呼び出し）
- **スクリプトとして実行**: 方法2（`if __name__ == "__main__":`）
- **柔軟な実行制御**: 方法3（コマンドライン引数）または方法4（フラグ）

最初は方法1と方法2を使いこなせれば十分です。慣れてきたら、プロジェクトの規模に応じて他の方法も試してみてください。

次のステップ

さらに学びたい方は、以下のトピックもおすすめです。

- ユニットテスト（unittest、pytest）の使い方
- デバッガ（pdb）の使い方
- ロギング（logging）モジュールの活用

コードを書く上で、「部分的に動かして確認する」スキルは非常に重要です。ぜひ実践で活用してみてください！