

globalキーワードの使い方

```
gbl = 3

print(f"1st global: {gbl}")

def change_global():
    global gbl
    gbl = 5
    print(f"changed_global: {gbl}")

def change_global2():
    # global gbl :
    # Pythonは最初に`gbl`がグローバル変数だと決める
    # なので、関数の一番上に書いた方が、流れるにも分かりやすい
    print(f"global: {gbl}") # エラーの根本原因
    global gbl # SyntaxError
    gbl = 7
    print(f"changed_global_again: {gbl}")

print(change_global())
print(change_global2())
# SyntaxError: name 'gbl' is used prior to global declaration
```

上記のコードを書いた私の意図

1. 最初に `gbl = 3` で初期化
2. `change_global()` が実行されて `gbl` が 5 に変更される
3. `change_global2()` が実行され、最初の `print` 文で 5 が表示されるはず
4. その後 `gbl` が 7 に変更される

どうして上記の考えが間違っているか！

①Pythonは関数内で`global gbl`という宣言を見ると、その関数内の`gbl`変数はグローバルスコープのものだと認識します。

②`global gbl`宣言の前に`gbl`を参照すると（例：`print(f"global: {gbl}")`）、Pythonはこの矛盾を検出し構文エラーを発生させます。

③これは`SyntaxError: name 'gbl' is used prior to global declaration`というエラーメッセージになります。

④そのため、`global`宣言は必ず変数を使用する前に書く必要があります。関数の最初の行である必要はありませんが、その変数の最初の使用よりも前に書く必要があります。

- **まとめ**

- Pythonは、関数全体を見て、`gbl`は、グローバル変数だと決める
- 構文的に、`global gbl`は、先に宣言すべきである（ルールである）
- しかし、`print(f"global: {gbl}")`は、その前に実行しようとしている
 - よって構文エラーとなる

訂正後のコード

```
gbl = 3

print(f"1st global: {gbl}")

def change_global():
    global gbl
    gbl = 5
    print(f"changed_global: {gbl}")

def change_global2():
    # global gbl
    # print(f"global: {gbl}")
    print("globalキーワードを使います")
    global gbl
    gbl = 7
    # `global gbl`は、宣言済み
    print(f"changed_global_again: {gbl}")

print(change_global())
print(change_global2())
# 1st global: 3
# changed_global: 5
# None
# globalキーワードを使います
# changed_global_again: 7
# None
```

- **まとめ2**

- 必ずしも、`global`宣言は、関数の1行目に書く必要はないが、**グローバル宣言をしてからグローバル変数を使わなければならない**

- ただ、ここがややこしいが、**グローバル変数を使うだけであれば、global宣言は、不要である**
 - さらにややこしいが、**global宣言をしたら、たとえ変更をしなくても、global変数は、宣言の後に使うこと！**
 - また、globalキーワードは、関数の1行目で書いた方が、勘違いが無くなると私は思います
-
- **補足：**上記のコードでNoneが返っている理由
 - Pythonの関数は必ず何かを返し、**return**文がないか値を指定しない場合は常にNoneを返します

END