1. Pythonのスコープ: 変数が定義される範囲

Pythonでは変数や名前がどこでアクセス可能か（スコープ）は、以下の4種類（***LEGBルール***）に分類されます。

① Local（ローカルスコープ）

- **対象:** 関数やメソッド内で定義された変数。
    - メソッドとは、class内の関数
        - class内へのアクセスは、インスタンス・クラス名 でアクセス
            - classの学習が別途必要です！
- **スコープの範囲:** 関数・メソッドの中だけ。ここで定義された値は外からアクセスできない。
    - ※同じ名前の変数が内外で定義されていても、ローカルスコープ内ではローカルの変数が優先。
- **例:**

```
def func():
    x = "local"  # ローカルスコープ
    print(x)

func()
print(x)  # NameError: x is not defined
```

---

② Enclosing（エンクロージングスコープ）

- **対象:** ネストされた関数（関数**内**関数）の「外側の関数スコープ」で定義される変数。
- **スコープの範囲:** 内側の関数から「外側の関数」にアクセス可能。
    - (ただし、エンクロージングスコープの値は「書き換え不可」。書き換えたい場合は `nonlocal` を使用)
        - アクセス出来るからと言って、変更できると思ってはいけないということ！（変更するには、nonlocalキーワードが必要）
            - さらに、nonlocalキーワードは、次に出てくる **globalキーワード**も同じですが、~~関数内の最初に書くべき~~です
                - ~~一つ目の理由は、分かりやすさ。二つ目の理由は、難しくて理解できませんでしたが、私は暗記することにしています！~~
- **例:**

```
def outer():
    x = "enclosing"  # エンクロージングスコープ
```

```
    def inner():
        print(x)  # 内側のスコープから、外側のスコープの変数にアクセス可能

    inner()

outer()
```

---

③ Global（グローバルスコープ）

- **対象:** 関数やクラスの外で定義された変数。
- **スコープの範囲:** スクリプト全体で利用可能。ただし、関数内で書き換えたい場合は `global` キーワードが必要。
- **例:**

```
x = "global"  # グローバルスコープ

def func():
    global x
    x = "modified"  # グローバル変数を編集
    print(x)

func()  # 出力: modified
print(x)  # 出力: modified
```

---

④ Built-in（ビルトインスコープ）

- **対象:** Pythonに組み込まれた「標準関数」「定数（`True` など）」などが事前に定義されたもの。
- **スコープの範囲:** どのスコープからもアクセス可能。
- **注意:** 自分のコードで同じ名前の変数を定義すると上書きされ、機能が失われる。
    - (でも、maxなど、よく普通の変数として使われている、学習教材の短いコードなら、
      書き換えられても、max()関数として使わない限り大丈夫でしょう)
- **例:**

```
len = 100  # ビルトイン関数 "len" を上書き（非推奨）
print(len([1, 2, 3]))  # TypeError: 'int' object is not callable
```

---

2. 変数探索の仕組み（LEGBルール）

Pythonは、上記4つのスコープを次の「**順序**」に従って変数を探索します：

1. **L (Local):** 現在の関数やメソッドのスコープ内で探す。

2. **E (Enclosing):** ネストされた関数の外側スコープ（エンクロージングスコープ）の中で探す。
3. **G (Global):** モジュール全体のスコープ（グローバルスコープ）を探す。
4. **B (Built-in):** Pythonが提供するビルトインスコープの中を探す。

- **重要なポイント:**
  - 変数は「最初に見つかったスコープ」で利用され、探索は終了します。
  - 見つからない場合は `NameError` が発生します。

---

3. 注意点

① グローバル変数の影響を理解する

- 関数内でグローバル変数を書き換える場合は、`global` キーワードが必要。
- 適切にスコープを分けることで、プログラムの予測可能性が高まる。

```
x = "global"

def func():
    x = "local"  # グローバル変数ではなく、ローカルスコープで新しい変数を定義
    print(x)

func()  # local
print(x)  # global （ローカル変数がグローバルには影響しない）
```

---

② 名前のオーバーライドには注意する

- ローカルスコープやグローバルスコープで定義した変数が、ビルトイン関数や他の変数を上書きしないように気をつける。
- 特にビルトイン関数（`len`, `list`, `str`など）は上書きするとエラーを引き起こします。

---

③ 再帰的なスコープ参照の回避

変数が複数スコープで定義されていると、意図せずに内側と外側で値が重複して混乱を招くことがあります。

```
x = 10  # グローバル変数

def outer():
    x = 20  # エンクロージングスコープ変数
    def inner():
        x = 30  # ローカル変数
        print(x)  # 最も内側のスコープ（ローカル）が参照される

    inner()
```

```
outer()  # 出力: 30
```

---

④ 繰り返しポイント

1. **変数は「探索される順序」で利用される（LEGBルール）。**
2. **スコープを意識すれば、変数の影響範囲を明確に限定できる。**
3. **`global` や `nonlocal` を意識すると、スコープの管理がスムーズになる。**
4. **ビルトイン関数（`len`, `print` など）は上書きしない。**

END