Python docstring完全ガイド

新人エンジニアが最初に学ぶべきドキュメント記法

📖 目次

---

docstringとは？

docstring（ドックストリング） = 関数やクラスの説明文

基本ルール

```python
def 関数名(引数):
    """ここに説明を書く（三重引用符で囲む）"""
    処理
```

なぜ必要？

- ✅ 他の人が読んでもわかる
- ✅ 未来の自分が読んでもわかる
- ✅ `help(関数名)` で説明が表示される
- ✅ チーム開発で必須

5大記法の比較

記法	特徴	使う場面	推奨度
PEP 257	最小限（1行）	個人開発	★★★☆☆
Google	読みやすい	チーム開発	★★★★★

記法	特徴	使う場面	推奨度
NumPy	詳細な表形式	データ分析	★★★★☆
reST	Sphinx用	公式ドキュメント	★★★☆☆
Epytext	古い（@記法）	保守のみ	★☆☆☆☆

迷ったら？

→ Googleスタイル一択！ 世界標準で最も読みやすい

各記法の詳細解説

1. PEP 257（最小限スタイル）

特徴: シンプル、1行で完結

❌ 悪い例

```python
def add(x, y):
    """calc"""
    return x + y
```

問題点:

- 何を計算するか不明
- 動詞がない
- ピリオドがない

✅ 良い例

```python
def add(x, y):
    """2つの数を足し算する。"""
    return x + y
```

改善点:

- 何をするか明確
- 動詞で始まる（「足し算する」）
- ピリオドで終わる（。）

---

2. Google（推奨スタイル）⭐

特徴: 読みやすい、書きやすい、Args/Returns形式

❌ 悪い例

python

```
def calculate(x, y, z=10):
    """
    計算する

    Args:
        x: 数
        y: 数
        z: 数
    """
    return x / y + z
```

問題点:

- 型情報がない
- 説明が不十分
- Returns がない
- デフォルト値の説明なし

✅ 良い例

python

```python
def calculate(x, y, z=10):
    """
    (x / y) + z を計算する。

    Args:
        x (float): 割られる数
        y (float): 割る数
        z (float, optional): 加算する数。デフォルトは10

    Returns:
        float: 計算結果

    Raises:
        ValueError: yが0の場合
    """
    if y == 0:
        raise ValueError("yは0以外")
    return x / y + z
```

改善点:

- ✅ 型情報がある（float）
- ✅ 説明が具体的
- ✅ Returns がある
- ✅ optional を明記
- ✅ Raises がある（エラー情報）

---

3. NumPy（データ分析向け）

特徴: 表形式、ハイフン使用、詳細に書ける

❌ 悪い例

```python
```

```
def calculate_mean(data):
    """
    統計計算

    Parameters
    ---
    data: リスト

    Returns
    --
    結果
    """
    return sum(data) / len(data)
```

問題点:

- ハイフンが少ない（3個）→ 8個以上必要
- 型がない
- 説明が不十分

✅ 良い例

```python
def calculate_mean(data):
    """
    リストの平均値を計算する。

    Parameters
    ----------
    data : list of float
        数値のリスト

    Returns
    -------
    float
        平均値

    Examples
    --------
    >>> calculate_mean([10, 20, 30])
    20.0
    """
    return sum(data) / len(data)
```

改善点:

- ✅ ハイフン8個以上（----------）
- ✅ 型がある（: 形式）
- ✅ 説明がインデント
- ✅ Examples がある

---

4. reST / Sphinx（公式ドキュメント向け）

特徴: :記法、各引数2行、Sphinx用

❌ 悪い例

```python
def format_user(name, age):
    """
    ユーザー情報

    :param name: 名前
    :param age: 年齢
    :return: 文字列
    """
    return f"{name}さん（{age}歳）"
```

問題点:

- :type がない
- 説明が不十分
- :rtype がない

✅ 良い例

```python
```

```
def format_user(name, age):
    """
    ユーザー情報を整形する。

    :param name: ユーザーの名前
    :type name: str
    :param age: ユーザーの年齢
    :type age: int
    :return: 整形された文字列
    :rtype: str

    .. note::
      ageは0以上の整数を想定
    """
    return f"{name}さん（{age}歳）"
```

改善点:

- ✅ :type がある
- ✅ 説明が具体的
- ✅ :rtype がある
- ✅ .. note:: がある

---

5. Epytext（非推奨）

特徴: @記法、古い、新規では使わない

なぜ非推奨？

- 古い記法（レガシー）
- 新規プロジェクトでは使わない
- 既存コードの保守でのみ見かける

参考例

```
python
```

```
def calculate_tax(price, tax):
    """
    税込み価格を計算する。

    @param price: 税抜き価格
    @type price: int
    @param tax: 税率
    @type tax: float
    @return: 税込み価格
    @rtype: int

    @deprecated: 新規プロジェクトではGoogleスタイルを使用してください
    """
    return int(price * (1 + tax))
```

新規開発では使わないこと！

実践：help()で確認する

ステップ1: 関数を定義

python

```
def my_function(name, age):
    """ユーザー情報を表示する。

    Args:
        name (str): ユーザーの名前
        age (int): ユーザーの年齢

    Returns:
        str: 整形された文字列
    """
    return f"{name}さんは{age}歳です"
```

ステップ2: helpで確認

python

```
help(my_function)
```

実行結果

```
Help on function my_function in module __main__:
```

```
my_function(name, age)
    ユーザー情報を表示する。

    Args:
        name (str): ユーザーの名前
        age (int): ユーザーの年齢

    Returns:
        str: 整形された文字列
```

書いたdocstringが整形されて表示される！

よくあるエラーと解決方法

エラー1: 引用符の数が間違い

python

```python
# ❌ 間違い
def wrong():
    "これは間違い"  # 1個だけ
    pass

# ✅ 正しい
def correct():
    """これは正しい"""  # 3個
    pass
```

エラー2: 位置が間違い

python

```python
# ❌ 間違い
"""これは間違い"""  # 関数の外
def wrong():
    pass

# ✅ 正しい
def correct():
    """これは正しい"""  # defの直後
    pass
```

エラー3: 型の書き方（Google）

```python
# ❌ 間違い
def wrong(x):
    """
    Args:
        x int: 説明 # カッコがない
    """
    pass

# ✅ 正しい
def correct(x):
    """
    Args:
        x (int): 説明 # カッコで囲む
    """
    pass
```

エラー4: ハイフンの数（NumPy）

```python
# ❌ 間違い
def wrong(x):
    """
    Parameters
    --- # 3個（少ない）
    """
    pass

# ✅ 正しい
def correct(x):
    """
    Parameters
    ---------- # 8個以上
    """
    pass
```

エラー5: UnicodeDecodeError（日本語の文字化け）

```python
```

```
# ❌ 問題が起きる
exec(open('file.py').read())

# ✅ 解決方法
exec(open('file.py', encoding='utf-8').read())
```

または ファイルの先頭に追加：

```
python

# -*- coding: utf-8 -*-
```

チェックリスト

良いdocstringの条件：

- [ ] 三重引用符（"""）で囲んでいる
- [ ] 概要が1行目にある
- [ ] 動詞で始まる（〜する）
- [ ] ピリオドで終わる
- [ ] 全引数を説明している
- [ ] 型情報がある
- [ ] 戻り値を説明している
- [ ] 例外を説明している（ある場合）
- [ ] デフォルト値を明記（ある場合）
- [ ] 使用例がある（複雑な場合）

まとめ

カイゼンのポイント

1. 型情報を必ず書く

```
python

# ❌ NG
x: 数

# ✅ OK
x (int): 整数
```

2. 説明は具体的に

```python
# ❌ NG
データ

# ✅ OK
ユーザーIDのリスト
```

3. 戻り値を書く

```python
# ❌ NG
（なし）

# ✅ OK
Returns:
  int: 合計値
```

4. 例外を書く

```python
# ❌ NG
（なし）

# ✅ OK
Raises:
  ValueError: xが負の場合
```

5. デフォルト値を明記

```python
# ❌ NG
z: 数

# ✅ OK
z (int, optional): デフォルトは10
```

新人エンジニアへのアドバイス

スタートは簡単に

1. **まずはGoogleスタイルから**始める

2. **help()で確認する習慣**をつける
3. **既存コードのスタイルに合わせる**
4. **完璧を求めすぎない**
5. **継続が大切**

実践方法

```python
# 1. 関数を書く
def my_calc(x, y):
    """2つの数を掛け算する。

    Args:
        x (int): 1つ目の数
        y (int): 2つ目の数

    Returns:
        int: x × y の結果
    """
    return x * y

# 2. helpで確認
help(my_calc)

# 3. 実行してテスト
print(my_calc(5, 3))  # 15
```

クイックリファレンス

Googleスタイル（コピペ用）

```python
```

```
def function_name(arg1, arg2, arg3=default):
    """1行で概要を書く。

    詳細な説明がある場合はここに書く。
    複数行でもOK。

    Args:
        arg1 (型): 引数1の説明
        arg2 (型): 引数2の説明
        arg3 (型, optional): 引数3の説明。デフォルトはdefault

    Returns:
        型: 戻り値の説明

    Raises:
        ExceptionType: エラーが起きる条件

    Examples:
        >>> function_name(1, 2)
        3
    """
    # 処理
    pass
```

参考リンク

- **PEP 257**: Python公式のdocstring規約
- **Googleスタイル**: 最も推奨される記法
- **NumPy/SciPyスタイル**: データ分析プロジェクト向け
- **Sphinx**: ドキュメント生成ツール

最後に

docstringは**コードを書く時間と同じくらい大切**です。

未来の自分、チームメイト、オープンソースのユーザーのために、 **わかりやすい説明を書く習慣**をつけましょう！

頑張ってください！ 🚀

この記事は新人エンジニアの実践的な学習体験から作成されました。