CM : README.md et documentation en Python

Python : README et documentation

présenté par

Revekka Kyriakoglou

le

15 mars 2026

Plan

Pourquoi documenter ?

Idée

Un projet **sans documentation** oblige le lecteur à deviner :

- **à quoi ça sert** (objectif),
- **comment l'installer** (dépendances),
- **comment l'utiliser** (commandes),
- **ce qui doit sortir** (résultats).

Pourquoi documenter ?

Idée

Un projet **sans documentation** oblige le lecteur à deviner :

- **à quoi ça sert** (objectif),
- **comment l'installer** (dépendances),
- **comment l'utiliser** (commandes),
- **ce qui doit sortir** (résultats).

Règle pratique : un collaborateur (ou vous-même dans 2 mois) doit pouvoir exécuter le projet en **2 minutes**.

Exercice éclair

Imaginez que vous recevez un dossier `projet/` avec `tp_projet.py`. Notez 3 questions immédiates :

1. ...
2. ...
3. ...

Exercice éclair

Imaginez que vous recevez un dossier `projet/` avec `tp_projet.py`. Notez 3 questions immédiates :

1. ...
2. ...
3. ...

Souvent : **comment lancer**, **quelles entrées**, **quelles sorties**.

README.md : le fichier d'accueil du projet

Définition

`README.md` est un fichier qui présente le projet. C'est la **première chose** qu'on lit pour comprendre :

- le but du programme,
- comment l'exécuter,
- ce que le programme produit.

README.md : le fichier d'accueil du projet

Définition

`README.md` est un fichier qui présente le projet. C'est la **première chose** qu'on lit pour comprendre :

- le but du programme,
- comment l'exécuter,
- ce que le programme produit.

Sur GitHub, le `README.md` s'affiche automatiquement sur la page du dépôt.

Pourquoi ça s'appelle README.md ?

Extension de fichier

Un nom de fichier a souvent une **extension** :

- `.py` : fichier Python
- `.txt` : texte brut
- `.pdf` : document PDF
- `.md` : **Markdown**

Pourquoi ça s'appelle README.md ?

Extension de fichier

Un nom de fichier a souvent une **extension** :

- `.py` : fichier Python
- `.txt` : texte brut
- `.pdf` : document PDF
- `.md` : **Markdown**

Markdown = une façon simple d'écrire du texte avec une mise en forme légère (titres, listes, code) **en restant lisible** même sans mise en forme.

Exemple : Markdown vs rendu

Ce que vous écrivez dans `README.md` :

```
# Mon projet
## Utilisation
python3 tp_projet.py
- crée results/tokens.csv
- crée results/vocab.csv
```

Exemple : Markdown vs rendu

Ce que vous écrivez dans `README.md` :

```
# Mon projet
## Utilisation
python3 tp_projet.py
- crée results/tokens.csv
- crée results/vocab.csv
```

Même dans un éditeur simple, ça reste lisible. Sur GitHub, c'est mis en forme automatiquement.

Le minimum (template) : 8 sections

Un README minimal (mais sérieux) :

1. **Titre** + 1 phrase (pitch)
2. **Description** (2–6 lignes)
3. **Prérequis** (Python, librairies)
4. **Installation** (si nécessaire)
5. **Utilisation** (commande(s) + exemple)
6. **Entrées** / **Sorties** (formats)
7. **Structure du projet** (arborescence)
8. **Tests** / **vérification** (checklist)

Le minimum (template) : 8 sections

Un README minimal (mais sérieux) :

1. **Titre** + 1 phrase (pitch)
2. **Description** (2–6 lignes)
3. **Prérequis** (Python, librairies)
4. **Installation** (si nécessaire)
5. **Utilisation** (commande(s) + exemple)
6. **Entrées** / **Sorties** (formats)
7. **Structure du projet** (arborescence)
8. **Tests** / **vérification** (checklist)

Si une section n'est pas utile, on peut la supprimer, mais l'**utilisation** est obligatoire.

Exemple court de README (projet corpus)

```
# Mini-analyseur de corpus (Token / Sentence / Corpus)

## Description
Lit des fichiers .txt, construit un corpus et produit
↪  des CSV
(tokens, phrases, vocabulaire).

## Prérequis
- Python 3.x
- pandas (optionnel)

## Utilisation
python3 tp_projet.py

## Entrées / Sorties
Entrée : dossier data/ avec des .txt
Sortie : dossier results/ avec tokens.csv,
↪  sentences.csv, vocab.csv
```

Exercice 1 : README minimal

Écrivez un `README.md` minimal pour votre projet :

1. un titre + 1 phrase,
2. une commande d'exécution,
3. la liste des fichiers CSV produits.

Exercice 1 : README minimal

Écrivez un `README.md` minimal pour votre projet :

1. un titre + 1 phrase,
2. une commande d'exécution,
3. la liste des fichiers CSV produits.

Objectif : votre voisin doit pouvoir exécuter sans vous poser de questions.

Markdown : 5 outils essentiels

````
# Titre
## Sous-titre

- liste
- liste

1. étape
2. étape

```python
# Add code here
```
[This is an external link to
↪  https://kyriakoglou.up8.site/algorithmiqueetprogrammation.
````

Markdown : 5 outils essentiels

````
# Titre
## Sous-titre

- liste
- liste

1. étape
2. étape

```python
# Add code here
```
[This is an external link to
↪  https://kyriakoglou.up8.site/algorithmiqueetprogrammation.
````

Pour le code : toujours un bloc `python ...` (copiable).

Exercice 2 : mise en forme

Transformez en Markdown :

- Mon projet s'appelle "AnalyseCorpus.py"
- Pour lancer : `python tp_projet.py`
- "Il crée tokens.csv et vocab.csv."

Faites un titre , puis une section `Utilisation`, etc.

Section `Utilisation` : le coeur du README

Ce qu'on attend

Une personne doit pouvoir :

1. installer (si besoin),
2. lancer une commande,
3. obtenir un résultat visible.

Mettez **au moins une commande** (pas juste du texte).

Exemple : utilisation

Idée

Dans un README, on peut expliquer comment lancer un projet.

```
## Utilisation

1) Mettre des fichiers .txt dans le dossier data/

2) Lancer dans un terminal :
python3 tp_projet.py

3) Vérifier les résultats dans le dossier results/
(ex : tokens.csv, sentences.csv, vocab.csv)
```

Docstring : documentation *dans* le code

Définition

Une **docstring** est un texte placé :

- au début d'un **module** (fichier),
- d'une **classe**,
- d'une **fonction** / **méthode**.

Elle sert à expliquer **ce que fait le code**.

Docstring : documentation *dans* le code

Définition

Une **docstring** est un texte placé :

- au début d'un **module** (fichier),
- d'une **classe**,
- d'une **fonction** / **méthode**.

Elle sert à expliquer **ce que fait le code**.

La docstring est accessible via `help(...)`.

Docstring de fonction : exemple

```python
def compter_mots(texte):
    """Compte les mots d'une chaîne.

    Args:
        texte (str): Une phrase ou un paragraphe.

    Returns:
        int: Nombre de mots (séparés par des espaces).
    """
    return len(texte.split())
```

Docstring de classe : exemple Token

```python
class Token:
    """Représente un token (mot ou symbole) d'une
    ↪  phrase.

    Attributes:
        forme (str): Forme du token.
    """

    def __init__(self, forme):
        """Initialise un Token.

        Args:
            forme (str): La forme du token.
        """
        self.forme = forme
```

Exercice 3 : écrire 2 docstrings

Ajoutez des docstrings :

1. à la classe `Sentence` (1–3 lignes : rôle + contenu),
2. à la méthode `frequences(self)` (Args/Returns).

Pensez : **Que fait la méthode ? Que renvoie-t-elle ?**

Lire la documentation : `help()`

Idée

Python peut afficher automatiquement la documentation (docstrings) avec `help()`.

```
# Dans l'interpréteur Python (ou dans un notebook) :

help(Token)                    # doc de la classe
help(Sentence.frequences)      # doc de la méthode
```

Lire la documentation : `help()`

Idée

Python peut afficher automatiquement la documentation (docstrings) avec `help()`.

```
# Dans l'interpréteur Python (ou dans un notebook) :

help(Token)                    # doc de la classe
help(Sentence.frequences)      # doc de la méthode
```

Si le résultat est vide ou pas clair : c'est un signal pour améliorer la docstring.

Générer une doc simple : `pydoc`

Commandes

- Voir la doc dans le terminal : `python -m pydoc tp_projet`
- Générer une page HTML : `python -m pydoc -w tp_projet`

Générer une doc simple : `pydoc`

Commandes

- Voir la doc dans le terminal : `python -m pydoc tp_projet`
- Générer une page HTML : `python -m pydoc -w tp_projet`

Le HTML généré utilise vos docstrings : c'est un bon test de qualité.

Exercice 4 : mini-vérification

1. Écrivez une docstring pour `Corpus.nb_tokens()`.
2. Vérifiez avec `help(Corpus.nb_tokens)`.
3. Générez `pydoc -w` et ouvrez le HTML.

Checklist (avant de rendre)

README :

- objectif en 2 lignes,
- commande(s) exécutable(s),
- entrées/sorties claires (`data/`, `results/`).

Docstrings :

- classe : rôle + attributs,
- méthode : ce que ça fait + ce que ça renvoie.

TP : README + docstrings

Livrables :

1. `README.md` (sections minimales + commande + entrées/sorties)
2. docstrings pour : `Token`, `Sentence`, `Corpus`
3. (option) dossier `docs/` avec HTML produit par `pydoc -w`

Objectif : projet compréhensible **sans ouvrir le code**.