# Titre

Clean Code en Python et gestion de code legacy

# Contenu validé

# Oui

# Catch phrase (2 lignes max)

Apprenez à organiser des projets Python faciles à faire évoluer. Ces techniques vous seront également utiles pour maintenir facilement des projets complexes.

# Description (10 lignes max)

Le langage Python est très expressif, entre autres grâce à son typage dynamique qui permet de faciliter et accélérer le développement de projets complexes. Cependant, cette liberté peut poser problèmes lors de la maintenance des applications et de l’ajout de nouvelles fonctionnalités.

Dans cette formation pragmatique, vous verrez les différentes techniques et bonnes pratiques, reconnues dans le domaine du développement logiciel (quel qu'en soit le langage), tout en mettant un accent particulier sur les codes en Python. Vous apprendrez ainsi les grands principes permettant de reconnaître les problèmes classiques (code smell) et comment les régler. Finalement, vous appliquerez ces techniques pour maintenir et faire évoluer des codes existants.

# 

# Identifiant technique

pycleancode

# Domaine

développement

# Sous-domaine

Python

# Url

/formations/developpement/python-clean-code-et-legacy

# Ordre dans la page

6

# Prochaines sessions

## Lyon

21-22 juin 2021

## Lille

14-15 juin 2021

## Paris

28-29 juin 2021

# Équilibre théorie / pratique

35% théorie / 65% pratique

# Public visé / participants

* Chef de projet en développement
* Développeur
* Testeur ayant une fibre développement
* Architecte
* Technical Leader

# Prérequis

* De bonnes connaissances de Python (aucun problème sur la syntaxe, connaissance de l’orienté objet)
* Connaissance du développement logiciel en général

# Durée

3j soit 21h

# Objectifs pédagogiques

* Identifier les principaux éléments qui forment le « Clean Code » quel que soit le langage et en particulier dans le cas de Python
* Analyser un code existant pour en critiquer son organisation
* Utiliser ces éléments sur un projet exemple, ainsi que sur du code existant
* Concevoir des codes faciles à faire évoluer
* Identifier l’intérêt d’un outil de festion de version

# Tarifs

1700 € HT par stagiaire (pauses et repas inclus)

# Plan de formation / programme

## Bonnes pratiques

### Formatage du code

### Conventions de nommage (PEP8)

### Concepts de DRY (Don’t Repeat Yourself) et KISS (Keep It Single Stupid)

### Gestion des paramètres dans les fonctions

### Gestion de la documentation en Python (via les ‘Docstring’)

### Annotation de types pour faciliter la revue de code et la mise en oeuvre des outils d'analyse statique

TP : ajouter de la documentation à un code Python, faire évoluer les nommages de variables d’un code existant

## Organisation d'un projet en python

### Notion de découpage en fichiers, en dossiers.

### Impact sur les ’import”

### Rôle des fichiers classiques (\_\_init\_\_.py, \_\_main\_\_.py)

### La structuration classique d’un fichier

## Utiliser les fonctions de python pour améliorer son code

### Utilisation des ‘fonctions spécifiques’ versus ‘fonctions builtins’ de la bibliothèque standard (Avantages / Inconvénients)

### Le meilleur code est pas de code

### Méthodes magiques

### Décorateurs

### Gestionnaire de contexte

TP : illustration des différentes techniques sur des cas simples

## Point sur la gestion de versions

### Intérêt d’un outil de gestion de versions

### Utilisation pour revenir en arrière et développer de nouvelles fonctionnalités

### Intérêt des messages de commit

### Parcours de l’historique

## Les principes en Programmation Orientée Objet

### Présentation des principes SOLID

* Single responsibility principle
* The open/closed principle
* Liskov's substitution principle
* Interface segregation
* Dependency inversion

### La loi de Demeter

### Reconnaître / écrire un code de qualité

TP : mise en œuvre sur un kata de programmation

## Les patrons de conception (design patterns)

### Le concept de design patterns et leur utilité

### Présentation de quelques design patterns connus et leurs cas d’utilisation classique

TP : Un cas d’usage implémentant le design pattern Observer, le design pattern Adapter, .

## Tests unitaires et refactoring

### Les principes de refactoring et de tests unitaires

### Les outils de tests unitaires (unittest, pytest) en python, notion de couverture de tests et génération de rapports

### Présentation du concept de développement piloté par les tests (TDD)

### L'automatisation des tests, l'agrégation de tests.

### Intégration dans un pipeline d’intégration continue (CI)

TP : mise en oeuvre sur un kata de programmation

## Maintenir un code existant (code legacy)

### Techniques de navigation dans un nouveau projet

### L’analyse statique du code et l’annotation de code

### Analyse dynamique du code avec “pycallgraph”, “sys.trace”

### Analyse des dépendances du code / gestion des versions

### Les techniques de refactoring sur du code existant (ajout de tests unitaires, notion de gold standard…)

TP : application des techniques vues en cours sur un kata de programmation classique

# Liens et ressources

<https://www.pearltrees.com/yvesdu/tests-jeux-d-essais/id28746238>

<https://github.com/thebjorn/pydeps>

Liste de katas : <https://kata-log.rocks/solid-principles>

Ressources pour compléter (code en PHP) :

* <https://alexsoyes.com/solid/>
* <https://alexsoyes.com/code-de-qualite/>
* <https://github.com/zedr/clean-code-python>
  + livre :<https://www.packtpub.com/product/clean-code-in-python/9781788835831>
  + code associé :<https://github.com/packtpublishing/clean-code-in-python>
* <https://www.octo.academy/fr/formation/204-clean-code>
* <https://www.pentalog.com/blog/clean-code-with-python>
* Patrons de conception : <https://www.toptal.com/python/python-design-patterns>