# Titre

Python idomatique

# Contenu validé

Oui

# Catch phrase (2 lignes max)

Utilisez enfin Python à sa pleine mesure ! Découvrez les bibliothèques internes et les mécanismes méconnus ou sous-exploités du langage qui vous permettront de gagner un temps précieux de mise en œuvre et apporteront plus de robustesse à vos applications.

# Description (10 lignes max)

Python est un langage puissant, orienté objet, convenant parfaitement à la majorité des développements informatiques. Outre sa lisibilité et la simplicité de sa syntaxe, un des grands avantages de Python est de s’appuyer sur une bibliothèque standard interne extrêmement fournie et des bibliothèques annexes nombreuses et variées. Ainsi, Python est aujourd’hui un des langages de prédilection dans de nombreux domaines exigeants : applications Web, calcul scientifique, scripts et traitements *backoffice*, analyse de données, *deep learning*, etc.

Cette formation présente concepts avancés et peu connus de Python : structures internes avancées, décorateurs, gestionnaires de contexte, itérateurs et leurs mises en œuvre respectives.

# Identifiant technique

pyidiomatique

# Domaine

développement

# Sous-domaine

Python

# Url

/formations/developpement/python-idiomatique

# Ordre dans la page

4

# 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

* développeurs confirmés
* débutant s’abstenir

# Prérequis

* bonne maîtrise des fondamentaux du langage Python
* pratique quasi quotidienne de ce langage.

# Durée

3 jours, 21 heures

# Objectifs pédagogiques

* S’approprier et s’orienter dans la grande diversité des bibliothèques internes de Python
* Apprendre à privilégier l’utilisation de ces bibliothèques afin de faciliter la maintenance de vos applications
* Maîtriser l’utilisation des techniques avancées du langage Python : Context Manager, Closures, Fonctions avancées, Itérateurs
* Choisir les bonnes structures de données en fonction de la complexité des cas d’usage

# Équilibre théorie / pratique

# 25% théorie / 75% pratique

# Tarifs

1650 € HT par stagiaire (pauses et repas inclus)

# Plan de formation / programme

## Rappels sur les bibliothèques utiles

### La bibliothèque « logging »

* Intérêt par rapport à la fonction print
* Intérêts et principes
* Mise en oeuvre d’un premier exemple (par programmation ou par fichier de configuration)
* Modifier les filtres
* Point sur les traces en multi-threading, en multi-processing

### Les arguments en ligne de commande

* La variable « sys.argv »
* La bibliothèque « argparse »
* La bibliothèque externe « click »

### Les fichiers de configuration

* La bibliothèque « configparser »
* La bibliothèque « json »
* Le traitement des fichiers YAML

TP : modifier un mini-projet fil rouge, pour lui ajouter un mécanisme de logs et une configuration à travers la ligne de commande et un fichier structuré (avec gestion des priorités de configuration)

## Les structures de données avancées et leurs outils

### Rappel sur les types en Python

* Les types avancés (« Containers », « Collections », « Sequences », « Mapping »)
* Propriétés des types et méthodes magiques (mutable, subscriptable, hashable, trucmuchable, …)
* Introduction aux annotations

### Programmation Orienté Objet Avancée

Accès aux paramètres et properties

\_\_getattr\_\_,

\_\_getattribute\_\_,

les 3 property

### La bibliothèque « collections »

* defaultdict,
* Counter
* deque
* namedtuple

### Les fonctions de manipulation des collections

* Fonctions « map »/ « filter » vs générateurs à la volée
* Fonctions “all” et “any”
* La bibliothèque « itertools », les itérateurs

### Les bibliothèques « heapq » et « bisect »

* Intérêts
* Exemple de mise en œuvre

TP : simplification de cas classiques en utilisant et justifiant la bonne structure de données / création d’une structure de données sur mesure (à l’aide des méthodes magiques)

## Techniques avancées

### Fonctions avancées et pièges classiques

* Le piège des paramètres par défaut de types mutables
* L’opérateur logique « or » lors des affectations
* Différentes façons de créer les fonctions (lambda, def)
* Espaces de noms et règle LEGB (variables « local », « non local », « global » et concept de closure)
* Paramètres à volonté avec \*args et \*\*kwargs
* Fonctions d’ordres supérieurs

### Les décorateurs (ou fonction englobante)

* Présentation du concept
* Les closures
* Cas d’usage des décorateurs
* Mise en œuvre (à l’aide de fonctions et de classes)
* Paramètres de décorateurs
* Études de décorateurs de la bibliothèque « functools » (@lru\_cache, @wraps)

### Les « Context Managers »

* Présentation et cas d’usage (fichiers, bases de données, réseau, accès concurrent aux données...)
* Intégration sur une classe avec les « dunder » méthodes spécifiques
* Intégration sur une fonction avec la bibliothèque « contextlib »