PYTHON & DATA : API, INTELLIGENCE ARTIFICIELLE

Python : POO et traitement de données

Approfondir Python avec la programmation orientée objet, la manipulation de fichiers (CSV, JSON), les modules standards et les bonnes pratiques de code.

DURÉE	TARIF HT	NIVEAU	LANGUE	GROUPE	FORMAT
3j (21h)	**890.00 €**	**Intermédiaire**	**Français**	**3-12**	**Formation**

1 PUBLIC VISÉ

- Développeurs Python maîtrisant les bases du langage souhaitant progresser vers un niveau professionnel.
- Personnes ayant suivi la formation Python : les fondamentaux du langage ou disposant d'une expérience Python équivalente.
- Développeurs autodidactes voulant structurer leur code Python avec la POO et les bonnes pratiques professionnelles.
- Data analysts ou ingénieurs souhaitant automatiser le traitement de fichiers CSV, JSON et XML avec Python.
- Personnes souhaitant poser les bases avant de suivre la formation pytest complète

2 PRÉREQUIS

- Bonne maîtrise des bases de Python : variables, fonctions, listes, dictionnaires, boucles et gestion des exceptions.
- La formation Python : les fondamentaux du langage est recommandée ou une expérience équivalente.
- Savoir utiliser le terminal et VS Code.

3 OBJECTIFS PÉDAGOGIQUES

- Concevoir et utiliser des classes Python : propriétés, méthodes, héritage et polymorphisme
- Maîtriser les méthodes spéciales (dunder) pour personnaliser le comportement des objets
- Lire, écrire et transformer des fichiers de données : CSV, JSON et XML
- Utiliser les modules standards essentiels : os, pathlib, datetime, logging
- Comprendre et écrire des décorateurs et des context managers
- Exploiter les générateurs et les compréhensions avancées pour un code efficace
- Écrire des tests unitaires avec pytest pour fiabiliser son code
- Structurer un projet Python professionnel avec packaging et gestion des dépendances

4 PROGRAMME DÉTAILLÉ

Jour 1 (7h) – Programmation orientée objet en Python

Module 1 – Classes et objets (3h)

Les bases de la POO en Python (1h30)

- Pourquoi la POO : organiser, réutiliser, maintenir
- Déclarer une classe et instancier un objet
- Le constructeur __init__ et l'attribut self
- Attributs d'instance et attributs de classe
- Méthodes d'instance, méthodes de classe (@classmethod) et méthodes statiques (@staticmethod)
- Cas pratique : classe Produit avec prix, stock et méthodes métier

Propriétés et méthodes spéciales (1h30)

- Les propriétés avec @property : getters et setters pythoniques
- Les méthodes dunder essentielles : __str__, __repr__, __len__, __eq__, __lt__
- __enter__ et __exit__ : implémenter un context manager
- __iter__ et __next__ : rendre un objet itérable
- Cas pratique : classe Panier avec représentation, comparaison et itération

Module 2 – Héritage et polymorphisme (2h)

Héritage (1h)

- Étendre une classe avec l'héritage : super() et surcharge de méthodes
- Héritage multiple en Python : la MRO (Method Resolution Order)
- Les classes abstraites avec abc.ABC et @abstractmethod
- Cas pratique : hiérarchie de classes pour des types de comptes bancaires

Polymorphisme et duck typing (1h)

- Le polymorphisme en Python : même interface, comportements différents
- Le duck typing : pourquoi Python favorise le comportement sur le type
- Les protocoles Python : Iterable, Comparable, Hashable
- isinstance et issubclass : vérifier les types sans rigidité
- Cas pratique : système de sérialisation polymorphe pour plusieurs formats

Module 3 – Décorateurs et générateurs (2h)

Décorateurs (1h)

- Les fonctions comme objets de première classe en Python
- Écrire un décorateur simple : wrapper de fonction
- Décorateurs avec arguments et functools.wraps
- Cas d'usage courants : mesure du temps d'exécution, cache, logging, retry
- Cas pratique : décorateur @timer et décorateur @retry pour les appels réseau

Générateurs et compréhensions avancées (1h)

- Les générateurs : yield et le protocole itérateur
- yield from : déléguer à un sous-générateur
- Expressions génératrices : économiser la mémoire sur de grandes collections

Compréhensions de dictionnaires et d'ensembles

- itertools : combinations, chain, groupby, islice
- Cas pratique : pipeline de traitement de données avec des générateurs

Jour 2 (7h) – Fichiers, modules et bonnes pratiques

Module 4 – Manipulation de fichiers de données (3h)

Fichiers CSV (1h)

- Lire un fichier CSV avec le module csv : reader et DictReader
- Écrire un fichier CSV avec writer et DictWriter
- Gérer les encodages, les délimiteurs et les guillemets
- Cas pratique : importer, filtrer et exporter un catalogue produits en CSV

Fichiers JSON (1h)

- Lire et écrire du JSON avec le module json : load, dump, loads, dumps
- Sérialiser des objets Python personnalisés avec un JSONEncoder
- Gérer les fichiers JSON volumineux avec un streaming parser
- Cas pratique : API mock locale avec lecture et écriture de fichiers JSON

Fichiers XML et autres formats (1h)

- Parser du XML avec ElementTree : find, findall, iter
- Créer et modifier un document XML
- Lire des fichiers .env avec python-dotenv
- Lire des fichiers de configuration TOML et INI
- Cas pratique : convertir un flux XML vers JSON

Module 5 – Modules standards essentiels (2h)

Gestion des fichiers et du système (1h)

- pathlib.Path : manipuler les chemins de manière orientée objet
- os et shutil : opérations sur les fichiers et dossiers
- glob et fnmatch : lister des fichiers selon un motif
- tempfile : créer des fichiers et dossiers temporaires
- Cas pratique : script d'organisation automatique de fichiers par extension

Dates, logging et variables d'environnement (1h)

- datetime : créer, formater, comparer et calculer des dates
- timezone : gérer les fuseaux horaires avec zoneinfo
- logging : niveaux de log, handlers, formatters et configuration
- os.environ et python-dotenv : gérer les variables d'environnement
- Cas pratique : script de traitement avec logs structurés et gestion des dates

Module 6 – Tests avec pytest (2h)

Premiers pas avec pytest (1h)

- Pourquoi tester : fiabilité, refactoring serein, documentation vivante
- Installer pytest et écrire son premier test
- Les assertions pytest : assert simple et messages d'erreur expressifs
- Organiser ses tests : fichiers, classes, nommage

Lancer les tests et lire les rapports

Tester des cas complexes (1h)

- Les fixtures pytest : préparer et nettoyer l'état des tests
- @pytest.mark.parametrize : tester plusieurs cas avec un seul test
- Mocker des dépendances avec unittest.mock et pytest-mock
- Mesurer la couverture de code avec pytest-cov
- Cas pratique : suite de tests complète pour les classes développées en jour 1

Pour aller plus loin : Formation pytest : tester son code Python en profondeur

Jour 3 (7h) – Architecture de projet et projet de synthèse

Module 7 – Structurer un projet Python professionnel (3h)

Organisation et packaging (1h30)

- Structure standard d'un projet Python : src/, tests/, docs/
- Les fichiers essentiels : pyproject.toml, README.md, .gitignore
- Créer un package Python installable avec pip install -e .
- Gérer les dépendances avec pip et requirements.txt
- Les environnements virtuels : venv et pourquoi les utiliser systématiquement

Bonnes pratiques et outillage (1h30)

- PEP 8 : les conventions de style Python
- Black : formater le code automatiquement
- Ruff : linter Python rapide pour détecter les erreurs
- Les type hints Python : annoter les fonctions et les classes
- mypy : vérification statique des types
- Cas pratique : configurer le pipeline qualité complet sur le projet

Module 8 – Projet de synthèse (4h)

Réalisation d'un outil Python orienté objet complet

- Cahier des charges : outil de traitement et d'analyse de données depuis des fichiers CSV et JSON
- Fonctionnalités : modélisation des données en classes avec héritage, lecture et transformation de fichiers CSV et JSON, export des résultats dans plusieurs formats, logs structurés avec le module logging, tests pytest avec fixtures et parametrize, structure de projet professionnelle avec pyproject.toml
- Étape 1 : modélisation des classes et des structures de données
- Étape 2 : lecture, transformation et export des fichiers
- Étape 3 : tests unitaires et mesure de couverture
- Étape 4 : packaging et configuration qualité
- Revue de code collective : architecture POO, qualité des tests, lisibilité
- Retour formateur individualisé sur le projet rendu

5 COMPÉTENCES VISÉES

- Concevoir une hiérarchie de classes adaptée à un problème métier en Python
- Lire, transformer et écrire des fichiers de données structurées (CSV, JSON, XML)
- Écrire du code Python idiomatique : décorateurs, générateurs, compréhensions
- Tester son code avec pytest et mesurer la couverture de tests
- Structurer et packager un projet Python réutilisable et maintenable
- Journaliser les événements d'une application avec le module logging

6 MODALITÉS PÉDAGOGIQUES

Formation délivrée en présentiel ou distanciel (visioconférence). Le formateur alterne entre méthode démonstrative (live coding commenté et progressif), méthode interrogative (analyse critique de code existant et discussion des choix de conception) et méthode active (exercices pratiques et mini-projets par module). L'accent est mis sur l'écriture de code Python idiomatique et professionnel plutôt que sur la mémorisation de l'API.

7 MOYENS PÉDAGOGIQUES ET TECHNIQUES

- Support de cours numérique mis à disposition des apprenants
- Dépôt GitHub de démonstration avec exercices et corrections par module
- Environnement de développement : VS Code + Python 3.12+ + pytest
- Pour le distanciel : visioconférence (Zoom ou équivalent), partage d'écran, chat en direct
- Accès à la plateforme pédagogique LaPolaris (supports, ressources, émargement)

8 MODALITÉS D'ÉVALUATION

- En cours de formation : exercices pratiques et mini-projets corrigés à chaque module
- En fin de formation : réalisation d'un outil Python complet avec POO, traitement de fichiers et tests
- Questionnaire d'auto-évaluation des acquis en fin de parcours

9 CRITÈRES D'ÉVALUATION

- Conception correcte d'une hiérarchie de classes adaptée au problème avec héritage et méthodes spéciales
- Lecture et transformation fiables de fichiers CSV et JSON avec gestion des erreurs
- Écriture de décorateurs fonctionnels réutilisables dans un contexte professionnel
- Suite de tests pytest couvrant les cas nominaux et les cas limites des classes critiques
- Structure de projet professionnelle avec pyproject.toml, environnement virtuel et outillage qualité configuré

10 MODALITÉS DE VALIDATION

Attestation de fin de formation délivrée à l'issue du parcours, conditionnée à une assiduité d'au moins 80 % et à la réalisation du projet de synthèse. L'attestation précise les objectifs atteints et les compétences acquises.

11 SUIVI ET ACCOMPAGNEMENT

- Feuilles d'émargement signées par demi-journée (présentiel) ou émargement numérique (distanciel)
- Traçabilité des activités pédagogiques réalisées
- Attestation d'assiduité délivrée en fin de formation
- Suivi individuel via les exercices corrigés et le projet de synthèse

12 CONDITIONS D'ACCÈS

Formation accessible sur inscription directe, sans prérequis administratif particulier. Le financement peut être pris en charge par l'employeur dans le cadre d'un plan de développement des compétences, ou en autofinancement avec possibilité de paiement en plusieurs fois.

13 DÉLAIS D'ACCÈS

Inscription possible jusqu'à **5 jours ouvrés** avant le début de la session. Pour toute demande urgente, nous contacter directement.

ACCESSIBILITÉ · HANDICAP

Nos formations sont accessibles aux personnes en situation de handicap. Pour toute situation nécessitant un aménagement (matériel, temporel ou pédagogique), nous vous invitons à nous contacter avant l'inscription afin d'étudier les adaptations possibles. Référent handicap : contact@lapolaris.fr

LaPolaris

TÉL. +33762584798

EMAIL contact@lapolaris.fr

WEB lapolaris.fr