Smartfuture
Certifications | Training | Technology solutions

FORMATION & CERTIFICATION • PCPP-32-201 (PCPP2)

PCPP2 – Certified Professional in Python Programming 2

Formation avancée en Python pour développeurs expérimentés. Maîtrisez la métaprogrammation, les design patterns, la concurrence, les tests avancés et le développement d'applications robustes afin de réussir la certification professionnelle PCPP2 (PCPP-32-201).

5 jours 35 heures	**Examen** PCPP-32-201 (PCPP2)
Modalité Présentiel Distanciel	**Niveau** avancé

INSCRIPTION / RÉSERVATION

Je m'inscris maintenant

Centre d'examen Pearson VUE

Formateurs Certifiés

OBJECTIFS PÉDAGOGIQUES

- Approfondir la POO et l'architecture logicielle en Python.
- Maîtriser la métaprogrammation (décorateurs, métaclasses).
- Appliquer les principaux design patterns en Python.
- Concevoir, tester et déboguer des applications complexes (unitaires, intégration, TDD).
- Maîtriser la concurrence et la programmation réseau.
- Se préparer à la certification PCPP2.

PUBLIC CIBLE

- Développeurs Python confirmés visant un niveau professionnel.
- Ingénieurs logiciels, architectes, leads et DevOps utilisant Python.
- Professionnels IT travaillant sur des projets backend, d'automatisation ou de data engineering.
- Titulaires d'une première certification Python souhaitant poursuivre le parcours.

PRÉREQUIS

- Maîtrise des bases de Python (structures, fonctions, POO, modules).
- Pratique régulière du développement logiciel.
- Idéalement : certifications PCAP et PCPP1 ou niveau équivalent.

PROGRAMME DE LA FORMATION – DÉTAILLÉ

Rappels avancés de Python & architecture logicielle

Rappels POO avancés

- Rappels sur les classes, instances, héritage, encapsulation, polymorphisme.
- Héritage multiple, MRO (Method Resolution Order) et bonne utilisation.
- Attributs de classe vs attributs d'instance, propriétés et descripteurs.
- Utilisation des dataclasses, slots, enums, NamedTuple pour des modèles robustes.

Architecture et structuration d'un projet Python

- Organisation en packages, modules, espaces de noms et imports.
- Architecture en couches (présentation, logique métier, accès aux données).
- Gestion des dépendances, virtualenv, pip, introduction à poetry/pipenv.
- Notions de SOLID, séparation des responsabilités, couplage et cohésion.

Métaprogrammation, introspection et décorateurs

Fonctions de première classe & closures

- Rappels sur les fonctions comme objets, passage de fonctions en paramètres.
- Portée des variables, closures, factory functions.

Décorateurs en profondeur

- Syntaxe des décorateurs, décorateurs de fonctions et de méthodes.
- Décorateurs paramétrés, empilement de décorateurs.
- Utilisation des décorateurs pour le logging, la mesure de performance, le caching, le contrôle d'accès.
- Introduction aux décorateurs de classes et cas d'usage.

Introspection et métaclasses

- Inspection d'objets au runtime : dir(), type(), isinstance(), issubclass(), hasattr(), getattr().
- Comprendre le rôle de type et le cycle de vie d'une classe.
- Création de métaclasses simples pour contrôler la création de classes.
- Cas d'usage : enregistrement automatique, validation, singletons, plugins, ORM.

PROGRAMME DE LA FORMATION – DÉTAILLÉ

Design patterns en Python

Principes de conception

- Rappels sur SOLID et principes d'architecture orientée objet.
- Patrons de conception : pourquoi et quand les utiliser.

Patrons de création

- Singleton : implémentations et pièges à éviter.
- Factory Method et Abstract Factory pour la création d'objets.
- Builder pattern pour la construction d'objets complexes.

Patrons structurels

- Façade : simplifier une API complexe.
- Adapter, Proxy, Decorator (OO) : enrichir ou adapter le comportement d'objets.
- Composite pour représenter des structures hiérarchiques.

Patrons comportementaux

- Observer / Pub-Sub : notifications et événements.
- Strategy : encapsulation d'algorithmes et sélection dynamique.
- Command : encapsuler des actions, annulation, historique.
- Template Method, State, Chain of Responsibility.

Tests avancés, qualité et débogage

Tests unitaires, d'intégration et TDD

- Rappels sur unittest et introduction à pytest.
- Organisation d'une arborescence de tests, conventions de nommage.
- Fixtures, parametrized tests, coverage.
- Mocks, fakes, stubs, monkeypatching.
- Test-driven development (TDD) sur une fonctionnalité complète.

Débogage et journalisation

- Utilisation de pdb, debug intégré des IDE, points d'arrêt.
- Module logging, configuration de loggers, handlers, formatters.
- Stratégies de logs pour la production (rotation, niveaux, corrélation)

PROGRAMME DE LA FORMATION – DÉTAILLÉ

Qualité de code & typage

- PEP 8, PEP 20, conventions de style.
- Typing avec annotations, mypy, pyright.
- Linting et formatters : flake8, pylint, black, isort.

Concurrence, parallélisme et asyncio

Concurrence avec threads

- GIL (Global Interpreter Lock) et implications sur le multithreading.
- Threading : création, synchronisation (Locks, RLocks, Events, Queues).
- Cas d'usage : I/O bound, tâches parallèles, worker threads.

Parallélisme avec multiprocessing

- Processus vs threads, isolation mémoire et communication inter-processus.
- Pool de processus, map, apply, gestion des résultats.

Programmation asynchrone (asyncio)

- Coroutines, tasks, event loop.
- Mots-clés async / await, gather, shield, timeouts.
- Gestion d'I/O réseau asynchrones, clients HTTP asynchrones.
- Comparaison entre threads, processus et asyncio selon les scénarios.

Programmation réseau, services web et API

Programmation réseau bas-niveau

- Rappels sur TCP/UDP, sockets, client/serveur.
- Création d'un serveur TCP simple en Python.

Clients HTTP et API REST

- Utilisation de requests pour consommer des API HTTP.
- Gestion des erreurs, timeouts, retries, pagination.
- Authentification (tokens, OAuth2 basique, API keys).

Frameworks web (aperçu)

- Introduction à Flask / FastAPI : routes, vues, réponses JSON.
- Gestion de la configuration, logging, middlewares simples.

PROGRAMME DE LA FORMATION – DÉTAILLÉ

Persistance des données et intégration

Fichiers et formats d'échange

- Lecture/écriture de fichiers textes et binaires.
- SON, CSV, configuration (INI, YAML – aperçu).

Bases de données relationnelles

- Module sqlite3 : connexions, curseurs, requêtes, transactions.
- ORM (aperçu) : SQLAlchemy, mapping objets–relations.

Intégration et services

- Scripts d'intégration : automatisation de traitements, ETL simples.
- Interaction avec des services externes (messageries, stockage cloud, etc.).

Packaging, déploiement et distribution

Packaging d'une application Python

- Structure d'un package, pyproject.toml, setup.cfg (aperçu).
- Scripts exécutables, entry points, console_scripts.

Distribution et déploiement

- Distribution via wheel, index privé / PyPI (concepts).
- Déploiement sur un serveur Linux (service systemd simple, virtualenv).
- Notions d'intégration continue (CI) pour tester et publier automatiquement.

Atelier de synthèse & préparation à l'examen PCPP2

Projet fil rouge

- Conception et implémentation d'une application complète utilisant plusieurs thèmes du cours.
- Mise en œuvre de design patterns, tests, logging, persistance, API, concurrence/asynchronisme.

Révision et examen blanc

- Cartographie du programme par rapport aux objectifs officiels PCPP2.
- Questions type examen, QCM et cas pratiques corrigés.
- Conseils pour l'inscription, la gestion du temps et la réussite de l'examen.