PYTHON & DATA : API, INTELLIGENCE ARTIFICIELLE

Tester son code Python avec pytest

Maîtriser pytest pour écrire des tests fiables, maintenables et automatisés en Python : tests unitaires, fixtures, mocking, couverture de code et intégration CI.

DURÉE	TARIF HT	NIVEAU	LANGUE	GROUPE	FORMAT
2j (14h)	**600.00 €**	**Intermédiaire**	**Français**	**3-12**	**Formation**

1 PUBLIC VISÉ

- Développeurs Python souhaitant structurer et fiabiliser leur code grâce aux tests automatisés.
- Personnes ayant suivi les formations Python : les fondamentaux du langage et Python : programmation orientée objet et traitement de données ou disposant d'une expérience Python équivalente.
- Développeurs back-end voulant intégrer les tests dans leur workflow quotidien et dans une pipeline CI/CD.
- Toute personne souhaitant adopter une approche professionnelle du développement Python avec pytest comme outil central.

2 PRÉREQUIS

- Bonne maîtrise de Python : fonctions, classes, exceptions, modules.
- Les formations Python : les fondamentaux du langage et Python : programmation orientée objet et traitement de données sont recommandées.
- Savoir utiliser le terminal et VS Code.

3 OBJECTIFS PÉDAGOGIQUES

- Comprendre les principes fondamentaux du test logiciel et le rôle de pytest dans l'écosystème Python
- Écrire des tests unitaires clairs et expressifs avec pytest
- Utiliser les fixtures pour gérer l'état et les dépendances des tests
- Mocker des dépendances externes avec pytest-mock et unittest.mock
- Paramétrer des tests pour couvrir de multiples cas sans duplication
- Mesurer et améliorer la couverture de code avec pytest-cov
- Organiser une suite de tests maintenable sur un projet réel
- Intégrer pytest dans une pipeline GitHub Actions

4 PROGRAMME DÉTAILLÉ

Jour 1 (7h) – Fondamentaux de pytest et fixtures

Module 1 – Introduction au test avec pytest (2h)

Pourquoi tester son code ? (1h)

- Les types de tests : unitaires, intégration, end-to-end
- pytest vs unittest vs doctest : positionnement et choix
- Installer pytest et structurer un projet de test : tests/, conftest.py, pytest.ini
- Conventions de nommage : fichiers test_*.py, fonctions test_*
- Lancer pytest : options essentielles -v, -s, -x, --tb
- Cas pratique : écrire les premiers tests sur un module de calcul simple

Assertions et gestion des erreurs (1h)

- Les assertions pytest : pas besoin de assertEqual, pytest réécrit les assertions
- Tester les exceptions attendues avec pytest.raises
- Tester les warnings avec pytest.warns
- Les messages d'assertion personnalisés
- Cas pratique : tester une classe métier avec cas nominaux et cas d'erreur

Module 2 – Fixtures (3h)

Les bases des fixtures (1h30)

- Le décorateur @pytest.fixture : créer une fixture réutilisable
- Injecter une fixture dans un test via le nom du paramètre
- Les scopes de fixture : function, class, module, session
- Les fixtures built-in de pytest : tmp_path, monkeypatch, capsys, caplog
- Cas pratique : fixture de base de données en mémoire avec SQLite

Fixtures avancées (1h30)

- Fixtures avec yield : setup et teardown dans une seule fonction
- Fixtures paramétrées : tester plusieurs configurations avec une seule fixture
- Dépendances entre fixtures : composer des fixtures complexes
- conftest.py : partager les fixtures entre plusieurs fichiers de test
- Cas pratique : fixtures pour un client HTTP simulé et un utilisateur authentifié

Module 3 – Paramétrage et organisation (2h)

Paramétrer les tests (1h)

- @pytest.mark.parametrize : tester N cas avec une seule fonction de test
- Paramétrage indirect : passer des paramètres via une fixture
- Combiner plusieurs @parametrize : produit cartésien des cas
- Nommer les cas pour un rapport lisible : ids
- Cas pratique : tester une fonction de validation avec une matrice de cas valides et invalides

Marqueurs et organisation (1h)

- Les marqueurs built-in : @pytest.mark.skip, @pytest.mark.skipif, @pytest.mark.xfail
- Les marqueurs personnalisés : @pytest.mark.slow, @pytest.mark.integration

Filtrer l'exécution par marqueur : pytest -m "not slow"

- Organiser les tests en classes TestXxx sans héritage obligatoire
- Cas pratique : marquer et filtrer les tests lents d'un projet

Jour 2 (7h) – Mocking, couverture et intégration CI

Module 4 – Mocking avec pytest-mock (3h)

Les bases du mocking (1h30)

- Pourquoi mocker : isoler le code testé de ses dépendances externes
- unittest.mock : Mock, MagicMock, patch
- pytest-mock : le mocker fixture, mocker.patch et mocker.MagicMock
- Mocker une fonction, une méthode, une propriété
- Vérifier les appels : assert_called_once_with, call_args_list
- Cas pratique : mocker un appel HTTP avec requests et vérifier le comportement

Mocking avancé (1h30)

- Mocker le système de fichiers avec tmp_path et monkeypatch
- Mocker les variables d'environnement avec monkeypatch.setenv
- Mocker le temps : datetime.now et time.sleep
- Les side_effect : simuler des exceptions ou des retours multiples
- Cas pratique : tester un service qui envoie des emails sans envoyer réellement

Module 5 – Couverture de code et qualité des tests (2h)

pytest-cov : mesurer la couverture (1h)

- Installer et configurer pytest-cov
- Générer un rapport de couverture : terminal, HTML, XML
- Lire un rapport HTML : identifier les branches non couvertes
- Les directives # pragma: no cover : exclure le code non testable
- Fixer un seuil de couverture minimum avec --cov-fail-under
- Cas pratique : analyser et améliorer la couverture d'un module existant

Qualité et maintenabilité des tests (1h)

- Le principe AAA : Arrange, Act, Assert
- Un test, une assertion de comportement : éviter les tests qui font tout
- Nommer les tests comme une documentation : test_should_raise_when
- Éviter les faux positifs : tester le bon comportement, pas l'implémentation
- Les anti-patterns courants : tests qui se dépendent, fixtures trop complexes

Module 6 – Intégration CI et projet de synthèse (2h)

pytest dans GitHub Actions (1h)

- Écrire un workflow GitHub Actions qui installe les dépendances et lance pytest
- Publier le rapport de couverture dans les artefacts du workflow
- Bloquer un merge si les tests échouent ou si la couverture passe sous le seuil
- Matrices de test : lancer les tests sur Python 3.11 et 3.12 en parallèle
- Cas pratique : pipeline CI complète pour un projet Python avec pytest et coverage

Module 7 – Projet de synthèse (1h)

Écriture d'une suite de tests complète sur un module Python fourni : fonctions utilitaires, classe de service avec dépendances externes, endpoint FastAPI ou script d'automatisation

- Couverture des cas nominaux, des cas limites et des cas d'erreur
- Rapport de couverture HTML produit et commenté
- Revue de code collective : lisibilité, isolation, pertinence des assertions
- Retour formateur individualisé sur les tests rendus

5 COMPÉTENCES VISÉES

- Écrire des tests unitaires et d'intégration robustes avec pytest
- Concevoir des fixtures réutilisables pour isoler les tests et gérer les dépendances
- Mocker efficacement les appels externes : API, base de données, système de fichiers
- Paramétrer les tests pour couvrir les cas nominaux, les cas limites et les cas d'erreur
- Analyser un rapport de couverture et identifier les zones non testées
- Intégrer l'exécution automatique des tests dans une pipeline CI/CD avec GitHub Actions

6 MODALITÉS PÉDAGOGIQUES

Formation délivrée en présentiel ou distanciel (visioconférence). Le formateur alterne entre méthode démonstrative (live coding sur des cas réels et progressifs), méthode interrogative (analyse des stratégies de test et des choix d'isolation) et méthode active (exercices pratiques et projet de synthèse). L'accent est mis sur l'écriture de tests immédiatement intégrables dans un projet professionnel existant.

7 MOYENS PÉDAGOGIQUES ET TECHNIQUES

- Support de cours numérique mis à disposition des apprenants
- Dépôt GitHub de démonstration avec les modules à tester et les corrections par étape
- 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 corrigés à chaque module
- En fin de formation : réalisation d'une suite de tests complète sur un module Python fourni
- Questionnaire d'auto-évaluation des acquis en fin de parcours

9 CRITÈRES D'ÉVALUATION

- Tests unitaires isolés couvrant les cas nominaux, les cas limites et les cas d'erreur
- Fixtures correctement scopées et réutilisables sans duplication entre fichiers de test
- Mocks ciblant les dépendances externes sans coupler les tests à l'implémentation
- Rapport de couverture HTML généré avec un taux atteignant le seuil défini
- Pipeline CI fonctionnelle qui bloque en cas d'échec ou de couverture insuffisante

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