PYTHON & DATA : API, INTELLIGENCE ARTIFICIELLE

Création d'une application IA avec Python et l'API OpenAI

Construire concrètement un chatbot ou un outil d'analyse de texte avec Python et l'API OpenAI. Prompting, streaming, structuration des sorties et intégration dans une API FastAPI : le prolongement naturel de votre formation FastAPI.

DURÉE	TARIF HT	NIVEAU	LANGUE	GROUPE	FORMAT
3j (21h)	~~900.00 €~~ **540.00 €**	**Intermédiaire**	**Français**	**1–6**	**Formation**

1 PUBLIC VISÉ

- Développeurs Python ayant suivi la formation *Créer une API REST avec Python et FastAPI* ou disposant d'une expérience équivalente avec FastAPI.
- Développeurs backend souhaitant intégrer des fonctionnalités IA (chatbot, analyse de texte, génération de contenu) dans leurs applications existantes.
- Freelances et développeurs en agence cherchant à enrichir leurs projets avec des appels à l'API OpenAI ou Anthropic.
- Profils techniques ayant déjà utilisé ChatGPT en mode "chat" et souhaitant passer à une intégration API réelle dans du code Python.

2 PRÉREQUIS

- Bonne maîtrise de Python : fonctions, dictionnaires, classes, gestion des exceptions, appels HTTP avec httpx ou requests.
- Avoir déjà créé au moins un endpoint FastAPI avec Pydantic : les formations *Python : POO et traitement de données* et *Créer une API REST avec Python et FastAPI* sont recommandées, ou une expérience équivalente.
- Notions de base sur les formats JSON et les variables d'environnement.
- Aucune connaissance en machine learning, en mathématiques ou en deep learning requise.

3 OBJECTIFS PÉDAGOGIQUES

- Comprendre le fonctionnement d'un LLM : tokens, contexte, température, rôles system/user/assistant
- Maîtriser les techniques de prompting essentielles : zero-shot, few-shot, chain-of-thought, instructions de format
- Consommer l'API OpenAI en Python avec gestion des erreurs, retry et streaming
- Structurer les sorties du modèle en JSON fiable grâce au function calling et au mode JSON
- Implémenter une conversation multi-tours avec gestion de l'historique et du contexte
- Intégrer un LLM dans une API FastAPI existante avec streaming SSE vers le frontend
- Construire un outil d'analyse de texte (classification, résumé, extraction d'entités) utilisable en production
- Estimer, surveiller et optimiser les coûts d'appels à l'API OpenAI

4 PROGRAMME DÉTAILLÉ

Jour 1 (7h) LLMs, prompting fondamental et structuration des sorties

Module 1 Comment fonctionne un LLM : ce que le développeur doit savoir (2h)

Modèles de langage et API OpenAI (1h)

- Qu'est-ce qu'un LLM : prédiction du token suivant, pas de "compréhension"
- La notion de contexte (context window) : taille, limites, ce que le modèle "voit"
- Les tokens : définition, comptage avec tiktoken, impact sur la facturation
- Température et top-p : contrôler la créativité vs le déterminisme
- Les modèles OpenAI disponibles : GPT-4o, GPT-4o-mini, o1 (quand utiliser lequel)
- Les hallucinations : pourquoi elles se produisent et comment les limiter

Cas pratique : observer l'effet de la température sur des sorties répétées (même prompt, cinq températures, analyse des variations)

Premier appel API en Python (1h)

- Créer sa clé API OpenAI, stocker dans .env, charger avec python-dotenv
- Installer et configurer le SDK officiel openai
- Structure d'un appel client.chat.completions.create() : modèle, messages, paramètres
- Lire et parser la réponse : choices[0].message.content, usage, finish_reason
- Gérer les erreurs API : RateLimitError, AuthenticationError, APIConnectionError
- Retry automatique avec backoff exponentiel : tenacity ou retry natif du SDK

Cas pratique : script Python autonome qui pose une question et affiche la réponse, le coût estimé et le finish_reason

Module 2 Techniques de prompting (3h)

Rôles et structure d'un prompt (1h)

- Le trio system / user / assistant : rôles et responsabilités
- Écrire un bon system prompt : persona, contraintes, format de sortie, ton
- L'ordre des instructions : ce qui est lu en premier vs en dernier
- Délimiteurs et structure : balises XML, triple backticks, numérotation
- Séparer les prompts du code : fichiers .txt, f-strings, variables d'environnement

Cas pratique : même tâche d'extraction, trois system prompts différents (analyser les écarts de qualité et de format)

Zero-shot, few-shot et chain-of-thought (1h)

- Zero-shot : formuler une instruction claire sans exemple
- Few-shot prompting : guider le modèle avec des exemples entrée/sortie bien construits
- Construire de bons exemples : diversité, cas limites, cohérence de format
- Chain-of-thought (CoT) : demander au modèle de "réfléchir à voix haute" avant de conclure
- Zero-shot CoT et son efficacité sur les tâches de raisonnement

Cas pratique : classification de tickets support (comparer zero-shot, few-shot (3 exemples) et CoT sur un jeu de 20 tickets réels)

Prompts pour tâches courantes (1h)

- Classification de texte : sentiment, intention, catégorie
- Résumé : longueur contrainte, format bullet, résumé exécutif
- Extraction d'entités : noms, dates, montants, références
- Transformation de contenu : reformulation, traduction, correction
- Génération de code : style, commentaires, gestion des cas limites

Cas pratique : pipeline de traitement d'emails (classification + extraction du nom de l'expéditeur + résumé en une phrase, tout en Python)

Module 3 Structurer les sorties du modèle (2h)

Obtenir du JSON fiable (1h)

- Pourquoi les LLMs produisent parfois du JSON invalide ou incomplet
- Instructions de format dans le prompt : schéma explicite, exemples annotés, contraintes négatives
- Le mode JSON d'OpenAI : response_format={ "type": "json_object" }
- Valider et parser la réponse : json.loads, gestion des exceptions, retry ciblé
- Pydantic pour valider la structure reçue : modèles, champs obligatoires, types

Cas pratique : extraire des données structurées (nom, montant, date, statut) depuis des factures textuelles non formatées

Function calling pour forcer un schéma de sortie (1h)

- Le function calling OpenAI : principe, paramètre tools, JSON Schema
- Différence entre appel de fonction et exécution : le LLM décide, Python exécute
- Déclarer un outil, intercepter tool_calls, retourner le résultat au modèle
- Forcer l'appel d'un outil avec tool_choice : utiliser le function calling comme extracteur de données

Cas pratique : extracteur d'informations de contact depuis des textes libres (retourne systématiquement un objet ContactInfo Pydantic valide)

Jour 2 (7h) Conversations, streaming et outil d'analyse de texte

Module 4 Conversations multi-tours et gestion du contexte (2h)

Construire et maintenir un historique (1h)

- Comment fonctionne la mémoire d'un LLM : il n'y en a pas (tout est dans messages[])
- Construire l'historique tour par tour : ajouter user et assistant à chaque échange
- Limites de la context window : détecter l'approche de la limite avec tiktoken
- Stratégies de troncature : supprimer les anciens tours, conserver le system prompt, résumer
- Résumé glissant : résumer l'historique ancien et l'injecter dans le system prompt

Cas pratique : chatbot de support technique en CLI (maintient le contexte, tronque intelligemment après 10 tours)

Injection dynamique de contexte (1h)

- Enrichir le system prompt à la volée : données utilisateur, catalogue produit, date/heure
- Templating de prompts : f-strings, Jinja2 pour les templates complexes

Séparer les prompts du code : fichiers .j2, constantes de configuration

- Versionner ses prompts comme du code source

Cas pratique : assistant qui adapte son comportement selon un profil utilisateur injecté (niveau technique, langue, secteur métier)

Module 5 Streaming et intégration FastAPI (3h)

Streaming avec l'API OpenAI (1h)

- Pourquoi streamer : UX, latence perçue, affichage progressif
- Activer le streaming : stream=True, itérer sur les ChatCompletionChunk
- Reconstituer la réponse complète depuis les deltas de tokens
- Détecter finish_reason et gérer les erreurs mid-stream

Cas pratique : générateur de lettres de motivation streamé dans le terminal, token par token

Endpoint FastAPI avec streaming SSE (1h30)

- Les Server-Sent Events (SSE) : protocole, format data: ...\n\n, connexion persistante
- Implémenter StreamingResponse dans FastAPI avec un générateur async
- Chaîner le stream OpenAI vers le stream SSE sans bufferiser la réponse complète
- Valider les entrées utilisateur avec Pydantic avant d'appeler le LLM
- Gérer le CORS pour une consommation depuis un frontend JavaScript
- Sécurité : ne jamais exposer la clé OpenAI côté client, architectures correctes

Cas pratique : endpoint POST /chat en FastAPI qui streame les réponses d'un assistant vers un client HTTP

Consommer le stream côté frontend (30min)

- Consommer un stream SSE en JavaScript avec fetch + ReadableStream
- Afficher les tokens au fur et à mesure dans le DOM
- Indicateur de chargement, bouton d'annulation avec AbortController

Cas pratique : interface chat HTML/JS minimaliste connectée à l'API FastAPI (réponses streamées, bouton "Stop")

Module 6 Outil d'analyse de texte (2h)

Concevoir un pipeline d'analyse (1h)

- Les tâches NLP couvables par un LLM : classification, résumé, extraction, scoring, comparaison
- Pipeline batch vs pipeline temps-réel : architectures et compromis
- Traitement parallèle : envoyer plusieurs appels en même temps avec asyncio.gather
- Utiliser l'API Batch OpenAI pour les workloads volumineux à moitié prix

Cas pratique : analyser 50 avis clients en parallèle (sentiment, score de 1 à 5, thème principal, action suggérée)

Endpoints d'analyse dans FastAPI (1h)

- Exposer plusieurs tâches d'analyse : POST /analyze/sentiment, /summarize, /extract
- Mutualiser la logique de prompting dans des fonctions réutilisables
- Cache des réponses : éviter de rappeler le LLM pour des entrées identiques

Logging des appels : stocker prompt, réponse, modèle, tokens consommés, durée

Cas pratique : API d'analyse de feedback utilisateur (trois endpoints, résultats JSON Pydantic, cache LRU intégré)

Jour 3 (7h) Coûts, robustesse et projet de synthèse

Module 7 Coûts, performance et fiabilité (2h)

Maîtriser et optimiser les coûts (1h)

- Comprendre la facturation OpenAI : input tokens vs output tokens, prix par modèle
- Calculer le coût d'un cas d'usage avant de développer : outil de simulation avec tiktoken
- Choisir le bon modèle selon la tâche : GPT-4o-mini suffit souvent, GPT-4o quand nécessaire
- Réduire les coûts : condenser les prompts, limiter max_tokens, utiliser le Prompt Caching
- L'API Batch OpenAI : 50% moins cher pour les traitements non temps-réel

Cas pratique : audit de coût d'un pipeline existant (identifier les gaspillages et proposer trois optimisations chiffrées)

Robustesse et monitoring (1h)

- Rate limits OpenAI : TPM, RPM, comment les monitorer et s'y adapter
- Retry avec backoff exponentiel : tenacity, délais progressifs, nombre max d'essais
- Timeouts et annulation : ne pas laisser une requête bloquée indéfiniment
- Évaluer la qualité des sorties : construire un jeu de tests, automatiser les évaluations
- Monitoring en production : logger latence, coût par appel, taux d'erreur

Cas pratique : middleware FastAPI qui logue automatiquement chaque appel LLM (prompt hash, tokens, durée, coût estimé)

Module 8 Atelier préparatoire au projet (1h)

- Choisir son sujet de projet parmi la liste proposée ou valider un sujet personnel
- Concevoir l'architecture : endpoints, prompts, gestion de l'historique, format de sortie
- Identifier les techniques à mobiliser : streaming, function calling, batch, cache
- Estimer le coût par appel et le coût mensuel projeté

Module 9 Projet de synthèse (4h)

Réalisation d'une application IA de bout en bout

- **Cahier des charges :** construire une fonctionnalité IA concrète au choix (chatbot de support, analyseur de contrat, générateur de tests unitaires depuis du code, résumeur de tickets, assistant de rédaction avec ton personnalisé)
- **Fonctionnalités attendues :** system prompt structuré et versionné, au moins une technique avancée (streaming, function calling ou batch), au moins un endpoint FastAPI avec validation Pydantic, estimation du coût par appel
- Étape 1 : conception du prompt et choix du modèle
- Étape 2 : implémentation du backend FastAPI
- Étape 3 : interface minimale (HTML/JS ou CLI) pour démontrer le fonctionnement
- Étape 4 : présentation (10 min) et revue collective (choix de prompting, gestion du contexte, coût)

Retour formateur individualisé sur les choix d'architecture et de prompting

5 COMPÉTENCES VISÉES

- Écrire des prompts efficaces et robustes adaptés à des tâches métier réelles
- Consommer l'API OpenAI en Python avec gestion des erreurs, retry et streaming
- Structurer les sorties LLM en JSON Pydantic valide via le function calling
- Implémenter une conversation multi-tours avec gestion de l'historique
- Intégrer un LLM dans une API FastAPI avec streaming SSE vers le frontend
- Estimer, surveiller et optimiser les coûts d'appels à l'API OpenAI

6 MODALITÉS PÉDAGOGIQUES

Formation délivrée en présentiel ou distanciel. Le formateur alterne entre méthode démonstrative (live coding avec l'API OpenAI sur des cas d'usage réels), méthode interrogative (analyse des sorties du modèle et discussion des choix de prompting) et méthode active (exercices itératifs et projet de synthèse fil rouge).

7 MOYENS PÉDAGOGIQUES ET TECHNIQUES

Formation délivrée en présentiel ou distanciel. Le formateur alterne entre méthode démonstrative (live coding avec l'API OpenAI sur des cas d'usage réels), méthode interrogative (analyse des sorties du modèle et discussion des choix de prompting) et méthode active (exercices itératifs et projet de synthèse fil rouge).

- Support de cours numérique mis à disposition
- Scripts Python annotés avec exercices et corrections par module
- Environnement : VS Code + Python 3.12+ + crédits API OpenAI fournis pour les exercices
- Pour le distanciel : visioconférence, partage d'écran, chat en direct

8 MODALITÉS D'ÉVALUATION

- En cours de formation : exercices de prompting corrigés et comparés à chaque module
- En fin de formation : réalisation d'une application LLM de bout en bout avec présentation
- Questionnaire d'auto-évaluation des acquis en fin de parcours

9 CRITÈRES D'ÉVALUATION

- En cours de formation : exercices de prompting corrigés et comparés à chaque module
- En fin de formation : réalisation d'une application LLM de bout en bout avec présentation
- Questionnaire d'auto-évaluation des acquis en fin de parcours

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 la vérification du dépôt GitHub de chaque apprenant en fin de parcours

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. LaPolaris est un organisme de formation en cours de certification Qualiopi.

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