Référentiel de compétences de la certification "Programmer et automatiser des tâches avec Python" (Tosa)

Modalités d'évaluation

Test numérique adaptatif Tosa.

- Typologies de questions : activités interactives (relier, glisser-déposer, cliquer sur zone, listes déroulantes, etc.), QCM et exercices de mises en situation rencontrées en entreprise via des exercices de manipulation d'un ou plusieurs logiciels dans leur environnement
- Format du test : 35 questions – 90 minutes , Questions réparties sur l'ensemble des 4 domaines.
- Algorithme : adaptatif (le niveau des questions s'adapte au niveau du candidat)
- Scoring mathématique IRT (Item Response Theory) : score sur 1000.
- Conditions de passage de la certification : en ligne, surveillance en présentiel par un examinateur ou à distance par proctoring

Critères d'évaluation

Chaque niveau de maîtrise est déterminé en fonction de la complexité de la tâche et du degré d'autonomie de l'individu qui réalise cette tâche.
Les niveaux sont catégorisés d'Opérationnel à Expert

Important : Pour les candidats visant la reconnaissance d'un socle de compétences professionnelles et l'obtention de la certification "Programmer et automatiser des tâches avec Python" (Tosa) , le seuil minimal de l'équivalent du niveau 3 (Opérationnel Tosa) est requis.

		Niveau Opérationnel (Score Tosa 551-725)	Niveau Avancé (726-875)	Niveau Expert (Score Tosa 876-1000)
Domaines	**Compétences**	**L'individu utilise Python efficacement pour une variété de tâches professionnelles. Il maîtrise des bibliothèques standards comme numpy, pandas, et matplotlib pour l'analyse de données et la visualisation. Il est capable de structurer des projets, d'utiliser des systèmes de contrôle de version et de déployer des applications. Il fait preuve d'autonomie et d'initiative dans la résolution de certains problèmes .**	**L'individu est très compétent avec Python dans toutes les situations professionnelles courantes. Il maîtrise des frameworks avancés tels que Django ou Flask pour le développement web, ou des bibliothèques de machine learning comme TensorFlow ou scikit-learn. Il peut optimiser le code pour l'efficacité et la performance et sait intégrer des systèmes Python dans des environnements de production. Il se sent à l'aise de travailler de manière autonome ou en équipe sur des projets complexes.**	**L'individu déploie des compétences expertes en Python dans des situations exigeant un haut niveau de spécialisation, telles que le développement de logiciels complexes, la gestion de grandes quantités de données ou l'implémentation d'algorithmes sophistiqués en intelligence artificielle. Il innove en utilisant Python pour créer des solutions originales à des problèmes nouveaux et complexes. De plus, il démontre une capacité à mentoriser d'autres développeurs, à diriger des équipes de développement et à influencer les meilleures pratiques au sein de sa profession.**

1. Langage et syntaxe	**1.1 Maîtriser la syntaxe de base et les structures de contrôle** Maîtriser la syntaxe de base de Python, incluant les boucles, les conditionnels et la gestion des erreurs, pour développer des programmes robustes et efficaces.	Gère les exceptions courantes en utilisant **try** et **except** pour prévenir les erreurs pendant l'exécution du code. Implémente des fonctions simples avec des arguments et des valeurs de retour pour organiser le code et améliorer sa réutilisabilité.	Utilise des compréhensions de listes pour créer des listes de manière plus concise et efficace. Intègre le contrôle de flux asynchrone pour gérer des opérations I/O sans bloquer l'exécution du programme.	Optimise les structures de contrôle pour maximiser l'efficacité et la clarté du code, notamment par le raffinement des conditions et l'amélioration de la logique des boucles. Développe des structures de contrôle complexes qui intègrent plusieurs niveaux de conditions, boucles imbriquées, et gestion avancée des erreurs pour des applications de grande envergure.
	1.2 Définir et utiliser des fonctions Créer des fonctions en Python, comprendre la portée des variables, gérer les arguments et retours, et utiliser les fonctions lambda pour simplifier le code et améliorer sa lisibilité.	Conçoit des fonctions lambda pour des opérations courtes et spécifiques sans nécessiter la définition formelle d'une fonction. Utilise les fonctions comme des objets de première classe, les passant en argument ou les utilisant comme valeurs de retour.	Développe des décorateurs pour modifier ou étendre le comportement des fonctions sans en changer le code source. Implémente des fonctions récursives pour traiter des structures de données complexes comme des listes imbriquées ou des arbres.	Optimise l'usage de fonctions en environnement de production, en évaluant leur performance et en améliorant leur efficacité. Conçoit des architectures logicielles qui utilisent des stratégies avancées de programmation fonctionnelle pour améliorer la modularité et la testabilité du code.
	1.3 Appliquer des concepts avancés de programmation Utiliser les décorateurs, générateurs, compréhensions de listes et techniques asynchrones pour écrire un code Python plus performant et concis.	Intègre le contrôle asynchrone dans des applications pour améliorer la performance en traitant les I/O de manière non-bloquante. Manipule les métaclasses pour changer le comportement de classes lors de leur création.	Conçoit des context managers pour gérer efficacement les ressources, comme les fichiers et les connexions réseau. Développe des protocoles de programmation asynchrone pour des applications web ou réseau complexes.	Innove en utilisant des fonctionnalités avancées du langage pour créer des solutions élégantes à des problèmes complexes. Contribue à des projets open source en proposant des améliorations sur le langage Python ou ses bibliothèques principales.
2. Structures de données et objets	**2.1 Manipuler des types de données primitifs et composés** Utiliser les listes, tuples, dictionnaires et ensembles, ainsi que les opérations courantes sur ces structures pour organiser et manipuler efficacement les données en Python.	Optimise l'accès aux données en utilisant des techniques comme le slicing de listes et le mapping de dictionnaires. Crée des classes pour encapsuler des structures de données complexes et leurs opérations associées.	Développe des structures de données personnalisées pour résoudre des problèmes spécifiques en utilisant des classes et des méthodes avancées. Intègre des algorithmes complexes pour optimiser le traitement et la manipulation des données.	Conçoit des systèmes pour gérer et analyser de grandes quantités de données de manière efficace. Innove dans la création de nouvelles structures de données qui améliorent significativement les performances des applications.
	2.2 Utiliser la programmation orientée objet Créer des classes, utiliser l'héritage et le polymorphisme, et appliquer des principes de conception avancée tels que les classes abstraites et les interfaces pour structurer des programmes orientés objet, facilitant la réutilisation et la maintenance du code.	Implémente le polymorphisme pour utiliser des interfaces communes pour différents types d'objets. Conçoit des classes abstraites pour définir des interfaces que d'autres classes doivent implémenter.	Développe des architectures logicielles qui intègrent plusieurs modèles de conception orientés objet pour résoudre des problèmes complexes. Utilise des patrons de conception avancés pour améliorer la flexibilité et la maintenabilité du code.	Innove dans la conception d'architectures orientées objet pour des systèmes à grande échelle, en intégrant des pratiques de design avancées et en optimisant les interactions entre objets. Mène des revues de code et des sessions de mentorat pour améliorer les compétences en conception orientée objet au sein de son équipe.

	2.3 Gérer et optimiser les données Manipuler les données de manière avancée avec des bibliothèques comme pandas et optimiser les structures de données pour améliorer les performances des programmes Python.	Package ses modules pour les distribuer et les réutiliser dans d'autres projets. Documente ses modules pour faciliter leur utilisation par d'autres développeurs.	Développe des packages complexes qui incluent plusieurs modules interdépendants. Utilise des techniques avancées pour la gestion de l'espace de noms et la résolution de dépendances au sein de ses packages.	Conçoit des frameworks modulaires utilisés par une large communauté de développeurs. Mène des projets de développement de logiciels qui s'appuient sur des architectures modulaires pour promouvoir la scalabilité et la flexibilité.
3.Modules et packages	**3.1 Utiliser et créer des modules** Importer des modules existants, créer de nouveaux modules et structurer le code en modules réutilisables pour favoriser la modularité et la maintenabilité en Python.	Package ses modules pour les distribuer et les réutiliser dans d'autres projets. Documente ses modules pour faciliter leur utilisation par d'autres développeurs.	Développe des packages complexes qui incluent plusieurs modules interdépendants. Utilise des techniques avancées pour la gestion de l'espace de noms et la résolution de dépendances au sein de ses packages.	Conçoit des frameworks modulaires utilisés par une large communauté de développeurs. Mène des projets de développement de logiciels qui s'appuient sur des architectures modulaires pour promouvoir la scalabilité et la flexibilité.
	3.2 Développer et distribuer des packages Créer et configurer des packages avec setuptools pour assurer leur distribution efficace.	Gère les versions de ses packages pour assurer la compatibilité et la stabilité à travers les différentes releases. Automatise les tests et le déploiement de ses packages pour garantir leur qualité avant la distribution.	Développe des stratégies pour la gestion des dépendances et la résolution de conflits au sein des environnements complexes. Intègre des outils de CI/CD pour automatiser la publication et la maintenance de ses packages.	Dirige des initiatives pour standardiser les pratiques de développement et de distribution de packages au sein de grandes communautés de développeurs. Innove en créant des outils et des méthodologies pour améliorer la distribution et la gestion de packages à grande échelle.
	3.3 Gérer les environnements et les dépendances Utiliser des environnements virtuels, gérer les dépendances avec pip et automatiser les configurations via Docker pour garantir des environnements de développement cohérents et reproductibles.	Utilise des outils comme Docker pour créer des environnements de développement et de production qui sont reproductibles et isolés. Automatise la configuration des environnements via des scripts ou des outils de gestion de configuration.	Intègre des systèmes de gestion de configuration pour automatiser et optimiser la création et la maintenance des environnements de développement et de test. Utilise des outils avancés comme Kubernetes pour gérer des environnements à grande échelle dans des déploiements de cloud.	Dirige des projets pour standardiser les environnements de développement à travers des organisations entières, en utilisant des pratiques de DevOps et des outils de gestion de conteneurs. Développe des stratégies pour assurer la cohérence, la sécurité, et l'efficacité des environnements de développement et de production dans des contextes multi-projets et multi-équipes.
	4.1 Analyser et profiler le code Utiliser des outils de mesure pour analyser la performance du code, identifier les goulots d'étranglement, et appliquer des méthodologies afin d'optimiser l'efficacité du programme.	Optimise le code en refactorisant les parties inefficaces identifiées lors du profiling. Utilise des outils de profiling en ligne pour surveiller les performances des applications en temps réel.	Intègre des pratiques de profiling et d'optimisation dans le cycle de vie du développement logiciel pour maintenir les performances tout au long de la production. Conseille sur les meilleures pratiques de profiling et d'optimisation au sein de son équipe ou de son organisation.	Développe des outils personnalisés pour analyser et améliorer les performances des systèmes complexes. Dirige des initiatives pour intégrer l'optimisation de performance dans les pratiques de développement standard.

4. Optimisation de code	**4.2 Améliorer l'efficacité du code** Appliquer des techniques pour réduire la complexité du code, optimiser les boucles et utiliser efficacement les ressources afin d'améliorer la performance globale des programmes.	Améliore la vitesse d'exécution en utilisant des algorithmes plus efficaces pour les tâches courantes. Applique des techniques d'optimisation comme le caching et la memorization pour réduire le coût des calculs répétitifs. Remplace les boucles inefficaces par des opérations de haut niveau plus efficaces, comme les compréhensions de listes.	Développe des stratégies pour minimiser la latence et maximiser le débit dans des applications critiques. Optimise des systèmes multi-thread ou multi-process pour exploiter pleinement les ressources matérielles.	Dirige des projets de refonte de systèmes pour intégrer les meilleures pratiques d'efficacité et de performance. Développe des frameworks et des bibliothèques qui améliorent l'efficacité des opérations de développement au sein de son industrie.
	4.3 Tester et déboguer Écrire des tests unitaires et d'intégration, utiliser des frameworks comme 'pytest' et appliquer des stratégies avancées de débogage pour garantir la qualité et la fiabilité du code.	Implémente des tests d'intégration pour vérifier que les différentes parties de son application travaillent ensemble correctement. Intègre des tests automatisés dans son processus de développement pour s'assurer que le code reste robuste face aux changements.	Conçoit des suites de tests pour couvrir des scénarios d'utilisation critiques et garantir la fiabilité de l'application sous diverses conditions. Optimise le processus de test pour réduire le temps d'exécution tout en maximisant la couverture du code.	Dirige des initiatives de qualité logicielle, en intégrant des pratiques de test avancées et des analyses statiques du code pour garantir la robustesse des applications. Peut former d'autres développeurs en techniques de débogage et en méthodologies de test pour améliorer la qualité du code au sein de son organisation