Formation Python

Perfectionnement

Votre formateur : Quentin BIDOLET

Nous contacter :
Christophe Guéroult
c.gueroult@2aiconcept.com
+33 6 56 85 84 33

0.1

Présentations

Quentin BIDOLET

- 5 ans d’expérience en développement Python
- Surtout du web, un profil plutôt Back avec un peu d’expérience sur du Front
- 2 ans d’expérience en formation

Participants

- Les bases de la programmation.
- Les concepts de la POO.
- Droits admin obligatoires sur tous les postes.
- Quelles sont vos attentes par rapport à cette formation ?
- Quelles sont les technos et langages sur lesquels vous travaillez habituellement ?

0.2

Go to Slack

Accès à Slack

- Vérifiez dans vos email que vous avez bien reçu le lien d’invitation au Slack de la formation.

- Merci de bien vouloir me rejoindre sur Slack.

0.3

GitHub

Le dépôt GitHub de la formation

- Donner le lien du dépôt de la formation sur Slack.
- A chaque notion abordée : un commit.
- Vous pourrez fork mon dépôt après la formation.
- Si vous souhaitez refaire les programmes après la formation, vous aurez accès à tous les commit pour chaque étape de l’application.

0.4

Déroulement de la formation

Horaires, temps de pause et dossier pédagogique

- De 9h à 17h ou de 9h30 à 17h30 (voir convocations).
- Une pause de 15mn le matin et une pause de 15mn l'après midi.
- Midi, une pause déjeuner de 1h15.
- Dernier jour de la formation se termine à 15h00 (voir convocations).
- Obligation du dernier jour à faire avant 15h donc, au retour de la pause déjeuner : vérifier les feuilles d'émargement et faire les évaluations formateur.
- Si certifications à passer, elles se feront en ligne le dernier jour l'après midi, après les évaluations formateur.

0.5

Installations outils et IDE

Installer Python

- PIP (Package Installer for Python) pour installer les librairies utilisées durant la formation https://pip.pypa.io/en/stable/installation/
- Python 3.10
  https://www.python.org/downloads/release/python-3913/
- Vérifiez dans cmd la version installée.
- https://docs.python.org/3/whatsnew/

Installer PyCharm

- https://www.jetbrains.com/pycharm/download/

0.7

Conventions

PEP 8

- 4 espaces par niveau d’indentation.
- Lignes de 79 / 120 caractères maximum.
- 2 lignes vides pour séparer les fonctions et classes.
- Méthodes dans une classe doivent être séparées par une ligne vide.
- UTF-8 sans déclaration d’encodage.
- https://peps.python.org/pep-0008/

1

Rappels

Ce qui est **Faux**

- Booléen **False**
- Null **None**
- L'entier zéro **0**
- Le réel zéro **0.0**
- Chaine de caractère vide **""**
- Liste vide **[ ]**
- Tuple vide **()**
- Dictionnaire vide **{ }**
- Set vide **set()**

- Tout le reste est **True**

L’import de module

- La façon la plus simple est d’utiliser **import module**, où module est le nom d’un autre fichier Python, sans préciser l’extension.
- On peut aussi importer en utilisant **from module import fonction**
- A vous de choisir selon votre préférence.
- Dans tous les cas vous pouvez utiliser l’alias **as**
- **dir()** pour lister tous les noms de fonctions ou de variables d’un module.

Nom des variables

- Un nom de variable peut contenir des **lettres minuscules**, **majuscules**, des **chiffres** et des **underscores** (_).

- Elles ne peuvent pas commencer par un nombre. Les noms commençant par un underscore sont traités spécialement, nous y reviendrons plus tard.

- Certains noms sont réservés, vous pouvez les consulter ci-dessous. https://www.w3schools.com/python/python_ref_keywords.asp

Commit

- Commit **codingstyle.py**
- Commit **builtinfuncs.py**
- Commit **docstrings.py**

2

Context managers

Gérer les ressources

Gérer les ressources est essentiel pour assurer un fonctionnement sécurisé de votre code. Cela inclut :

- **La libération des ressources** : Fichiers, sockets, connexion aux bases de données etc. Cela permet d'éviter les fuites de mémoires, blocages et problèmes de performance.

- **La gestion des erreurs** : Effectuer des opérations de nettoyage lorsqu'une exception est levée.

Try … finally

- Try-finally garantit l’exécution de certaines opérations même si une exception se produit à l’intérieur du bloc.

- Généralement ce sont des opérations de nettoyage, ce qui permet de s’assurer que les ressources sont correctement libérées.

Commit

- Commit **try-finally.py**

Le mot-clé with

- with est un mot-clé permettant de simplifier la gestion des ressources.
- Cela évite d'utiliser explicitement les blocs try-finally avec une syntaxe plus propre et lisible.

```
with open("example.txt", "r") as file:
    content = file.read()
```

- Dans cet exemple, le bloc de code ne s'exécute que si le fichier est correctement ouvert et est fermé automatiquement après le bloc.
- with ne fonctionne qu'avec des objets qui supportent certaines méthodes spécifiques.

Utilisation de with

- Ouverture et fermeture des fichiers.
- Gestion des verrous pour contrôler les accès aux ressources partagées.
- Gestion des transactions de base de données.

Commit

- Commit **with.py**

Context Managers

- Les context managers sont des objets qui définissent des méthodes pour être utilisé dans un bloc with.

- L’objectif est de fournir des méthodes pour effectuer des actions de nettoyage avant et après l’exécution d’un bloc de code.

Avantages de with

- Gestion automatique des ressources
- Gestion des erreurs et exceptions
- Simplification du code
- Réutilisation du code
- Extensibilité

Async et with

- async with permet d’utiliser des contexts managers asynchrones.
- Indispensable pour la gestion des ressources dans un environnement asynchrones

Créer son Context Manager

- Vous pouvez définir une classe avec les méthodes spéciales **__enter__()** et **__exit__()** .

- **__enter__()** : le code pour acquérir la ressource. Elle peut retourner l'objet lui-même, la ressource ou toute autre valeur utile. C'est la valeur qui sera assignée à la variable qui suit as.

- **__exit__(self, exc_type, exc_val, exc_tb)** : le code pour libérer la ressource. Les arguments exc_type, exc_val et exc_tb sont liés aux exceptions. S'il y en a une, ces variables peuvent être utilisées pour gérer les exceptions ou les propager. Si vous retourner True, vous ignorer l'exception et vous continuerez l'exécution. La plupart du temps, il ne faut rien retourner (None) ce qui va propager l'exception.

Commit

- Commit **context-manager-classe.py**

Context Manager par fonction

- On peut créer un context manager avec une fonction génératrice au lieu d'une classe.
- Pour cela il faut utiliser le décorateur **contextlib.contextmanager** fourni par le module **contextlib**.
- Les opérations d'acquisition et de libération sont séparées grâce à **yield.**
- Le code avant yield correspond à l'acquisition de la ressource (comme __enter__()).
- Le code après yield pour la libération des ressources (comme __exit__()).

Commit

- Commit **context-manager-fonction.py**

Context Manager asynchrone

- Pour une classe : il faut définir les méthodes spéciales **__senter__()** et **__sexit__()**, qui sont elles-mêmes asynchrones.
- La logique est la même que pour les fonctions __enter__() et __exit__().
- Pour une fonction : décorateur **contextlib.asynccontextmanager** fourni par le module **contextlib**.
- Ne pas oublier de définir notre fonction génératrice async.

Commit

- Commit **context-manager-async.py**

Conseils

- Gardez l’API du context manager simple.
- Documentez.
- Gérez et propagez les erreurs correctement.
- Pensez à la réutilisation.
- Pensez à l’extensibilité.
- Écrivez des tests unitaires et d’intégrations.

TP : CRUD

- L’objectif de ce TP est de créer un context manager pour gérer la connexion à une base de données SQLite et effectuer les opérations CRUD (Create, Read, Update, Delete) sur une table.

- Vous devez créer deux versions du context manager : une basée sur les classes et une basée sur les fonctions.

- Veuillez trouver sur Slack un exemple de code que doit exécuter votre context manager.

3

Meta-classes

Ancien mode vs nouveau mode

- **Ancien mode** : classe qui n'héritent pas explicitement d'un objet. Avec cette façon de faire, certaines fonctionnalités ne sont pas supportées comme les propriétés, descripteurs ou les méthodes spéciales pour les attributs.

- **Nouveau mode** : Héritent explicitement de l'objet. Toutes les fonctionnalités introduites avec Python 2.2 sont supportées.

  Python 3 : A partir de Python 3, toutes les classes sont considérées comme des classes "nouveau mode", par défaut, même si elles n'héritent pas explicitement de l'objet. Il n'y a plus de distinction entre l'ancien mode et le nouveau mode.
  Les métaclasses fonctionnent avec les classes du "nouveau mode" et on été introduites avec elles.

Qu’est ce qu’une méta-classe ?

- Une métaclasse est une classe dont les instances sont elles-mêmes des classes.
- Les métaclasses définissent comment les classes sont créées, en modifiant leur comportement, leur structure ou en ajoutant des fonctionnalités supplémentaires.
- Ce mécanisme repose sur le fait que tout est objet en Python, les classes y compris.

Typage et classes

- Le typage classe les objets Python selon leurs fonctions, nature ou le type d’opération qu’ils peuvent effectuer.

- Python est un langage à typage dynamique, c’est-à-dire que le type d’un objet est déterminé à l'exécution du programme.

- Les métaclasses permettent de contrôler la création et le comportement des classes. Par défaut, une classe est un objet instancié depuis la métaclasse “type”.

Définir une classe dynamiquement

- Créer une classe dynamiquement c’est créer la classe à l'exécution, sans déclarer explicitement la classe avec le mot-clé class.

- La fonction type() permet de créer une classe dynamiquement.

Commit

- Commit **creation-classe-dynamiquement-simple.py**
- Commit **creation-classe-dynamiquement-avancé.py**

Créer sa meta-classe

Il faut hériter de la métaclasse “type” et redéfinir ses méthodes, les plus importantes sont :

- **__new__**: Appelée lors de la création de la classe. Elle permet de modifier le nom de la classe, les bases et les attributs avant la création de la classe.
- **__init__**: Appelée après la création de la classe.
- **__call__**: Appelée lorsqu'une classe est instanciée. Vous pouvez modifier l'instance ici, si nécessaire.

Commit

- Commit **creation-metaclasse.py**

Cas d’utilisation

Vous pouvez utiliser les métaclasses pour :

- Valider les attributs de classe lors de son instanciation
- Modifier les attributs avant sa création, pour raison de compatibilité par exemple.
- Ajouter des méthodes ou des attributs à des classes. Cela peut réduire la duplication de code.
- Contrôle de l’instanciation, peut être utile lors de la conception de Proxy, Singleton ou Factory.

Inconvénients

- Attention, les métaclasses sont à utiliser avec modération.
- Cela augmente la complexité du code et le rend plus difficile à comprendre donc à maintenir.
- Attention à la compatibilité avec d'autres bibliothèques, si deux méta-classes différentes sont utilisés pour définir des classes de base, il peut être difficile de les faire fonctionner ensemble.

4

Fonctions avancés

Les fonctions internes

- Une fonction interne est une fonction définie à l'intérieur d'une autre fonction.
- Elles peuvent être utilisées pour encapsuler la logique spécifique à une fonction externe en la rendant inaccessible depuis l’extérieur.
- Cela permet aussi une meilleure organisation du code en regroupant des parties étroitement liées.

Commit

- Commit **functions.py**

Les bases des internes

- **Encapsulation** : Cela permet de masquer ces détails de l'extérieur, en rendant le code plus lisible et plus facile à maintenir. L'encapsulation aide à séparer les responsabilités et à éviter les fuites d'implémentation.

- **Conservation d'état** : Elles ont accès aux variables locales de la fonction englobante, même après la fin de l'exécution de celle-ci. Cela leur permet de se souvenir du contexte dans lequel elles ont été créées.

Inner vs privé

- Les méthodes avec un préfixe (par exemple, _private_method()) sont des méthodes "privées".

- La différence avec une fonction interne est la portée d'accès : les fonctions internes sont totalement inaccessibles depuis l'extérieur de la fonction englobante, tandis que les méthodes privées peuvent toujours être appelées, bien que la convention est de ne pas le faire.

- Les fonctions internes offrent un niveau d'encapsulation plus fort que les méthodes privées.

Closure

- Une fonction interne peut être utilisée comme une **closure**.
- C’est une fonction qui se souvient des valeurs dans le contexte où elle a été créée, même si elle n'est plus actuellement en ce contexte.

```
def outer_function(x):
    def inner_function(y):
        return x + y
    return inner_function

closure = outer_function(10)
print(closure(5))  # affiche 15
```

Cas d’utilisation d’une closure

- Création de compteurs, caches etc. qui nécessitent un état persistant entre les appels de fonctions.
- Pour des fonctions paramétrées qui nécessitent d’être configurées avec des paramètres spécifiques lors de leurs créations. Par exemple, une fonction multiplication qui utilise une variable spécifique de son contexte.
- Implémenter des décorateurs à l’intérieur d’une fonction.
- Pour gérer l’accès à des ressources partagées, comme des fichiers, sockets ou connexion à une base de données. On peut encapsuler l’ouverture, la fermeture et les détails de l’implémentation.
- Implémenter des callbacks dans les environnements asynchrones.

Commit

- Commit **closure.py**

5

Concurrence et parallélisme

Multithreading

- Un processus est une instance d’un programme informatique en cours d’exécution, un thread est une entité au sein d’un processus.
- Le multithreading est un programme qui appelle plusieurs threads.
- Cela peut considérablement accélérer la performance de votre programme même si cela peut amener des défis de coordination.

Cas d’utilisation

- Un serveur web va utiliser plusieurs threads pour traiter plusieurs requêtes en même temps.
- Un algorithme d'analyse d'image générera plusieurs threads à la fois et segmentera une image en quadrants pour appliquer un filtrage à l'image.
- Une application de ray-tracing lance plusieurs threads pour calculer les effets visuels tandis que le thread principal de l'interface graphique dessine les résultats finaux.

MultiProcessing vs MultiThreading 1/2

- Lorsqu'un ordinateur exécute une application, l'instance du programme qui s'exécute est appelée un processus.
- Un processus est constitué du code du programme, de ses données et d'informations sur son état.
- Chaque processus est indépendant et possède son propre espace d'adressage et sa propre mémoire. Un ordinateur peut avoir des centaines de processus actifs en même temps, et le travail d'un système d'exploitation est de les gérer tous.
- À l'intérieur de chaque processus, il existe un ou plusieurs sous-éléments plus petits appelés threads. Les threads sont les unités de base que le système d'exploitation gère, et il alloue du temps sur le processeur pour les exécuter réellement.

MultiProcessing vs MultiThreading 2/2

- Il est possible d'écrire des programmes parallèles qui utilisent plusieurs processus travaillant ensemble vers un objectif commun, ou des programmes qui utilisent plusieurs threads dans un seul processus.

- Les threads qui appartiennent à un même processus partagent l'espace d'adressage du processus, ce qui leur donne accès aux mêmes ressources et à la même mémoire, y compris le code exécutable et les données du programme.

- Mais cela peut aussi créer des problèmes si les threads ne coordonnent pas leurs actions.
  Le partage des ressources entre des processus distincts n'est pas aussi facile que le partage entre les threads d'un même processus.
  Parce que chaque processus existe dans son propre espace d'adressage.

Concurrence vs Exécution parallèle

Concurrence	Parallélisme
Structure d'un programme pouvant être exécutées dans un ordre quelconque.	Exécution simultanée de tâches sur différents processeurs.

Global Interpreter Lock 1/2

- L'interpréteur Python n'est pas thread-safe en interne, il a donc un interpréteur global.
- Lock (GIL), qui permet à un seul thread à la fois d'exécuter des bytecodes Python.
- C'est pourquoi un seul processus Python ne peut généralement pas utiliser plusieurs cœurs de processeur à la fois.
- Lorsque nous écrivons du code Python, nous n'avons aucun contrôle sur le GIL, mais une fonction intégrée ou une extension écrite en C peut libérer le GIL tout en exécutant beaucoup de tâches. En fait, une librairie Python codée en C peut gérer le GIL, lancer son propre OS threads et profiter de tous les cœurs de processeur disponibles. Cela complique le code de la bibliothèque considérablement, et la plupart des auteurs de bibliothèque ne le font pas.

Global Interpreter Lock 2/2

- Le GIL n'a pas beaucoup d'impact sur les performances des programmes multithreads liés aux E/S car le verrou est partagé entre les threads pendant qu'ils attendent des E/S.

- Mais un programme dont les threads sont entièrement liés au processeur, par exemple, un programme qui traite une image en plusieurs parties à l'aide de threads, ne deviendrait pas seulement un thread unique en raison du verrou, mais verra également une augmentation du temps d'exécution par rapport à un scénario où il a été écrit pour être entièrement monothread.

- C’est pourquoi les développeurs Python préfère généralement le Multiprocessing au MultiThreading.

Python Multithreading

- On utilise le module threading de Python pour créer plusieurs threads simultanés.
- https://docs.python.org/3/library/threading.html
- Pour avoir un aperçu sur l'utilisation des ressources, le gestionnaire des tâches peut être utilisé.

Commit

- Commit **multithreading.py**

Python Multiprocessing

- Pour tirer parti des processeurs multiples et réaliser une véritable exécution parallèle en Python, plutôt que de structurer notre programme pour utiliser plusieurs threads, nous devrons utiliser plusieurs processus.

- Le module multiprocessing de Python fournit une API pour la création de processus supplémentaires qui ressemble beaucoup au module threading.

Commit

- Commit **multiprocessing.py**

Le Scheduler

- Les threads ne s'exécutent pas simplement quand ils le veulent. Un ordinateur peut avoir des centaines de processus avec des milliers de threads qui veulent tous avoir leur tour pour fonctionner sur une poignée de processeurs.

- C'est le travail du système d'exploitation. Le système d'exploitation comprend un planificateur qui contrôle quand les différents threads et processus obtiennent leur tour d'exécution sur le CPU. Le scheduler (ou l'ordonnanceur) permet à plusieurs programmes de s'exécuter simultanément sur un seul processeur. Lorsqu'un processus est créé et prêt à être exécuté, il est chargé en mémoire et placé dans la file d'attente des processus prêts

Commit

- Commit **scheduler.py**

Data Race 1/2

- L'un des principaux défis de l'écriture de programmes concurrents consiste à identifier les dépendances possibles entre les threads pour s'assurer qu'ils n'interfèrent pas entre eux et ne causent pas de problèmes. Les Data Race sont un problème courant qui peut se produire lorsque deux ou plusieurs threads accèdent simultanément au même emplacement en mémoire;

- Au moins un de ces threads écrit à cet emplacement pour en modifier la valeur.

- On utilise les techniques de synchronisation pour protéger un programme contre les Data Race.

- Chaque fois que plusieurs threads lisent et écrivent simultanément une ressource partagée, cela crée un potentiel comportement incorrect, comme un effacement de données, mais cela peut être empêché en identifiant et en protégeant les sections critiques du code.

Data Race 2/2

- Une section critique ou une région critique est une partie d'un programme qui accède à une ressource partagée, telle qu'une mémoire structurée en données ou un périphérique externe, et qui peut ne pas fonctionner correctement si plusieurs threads y accèdent simultanément.

- Une incrémentation par exemple, est une section critique, parce que c’est une opération qui englobe trois sous opération: Lecture, Modification, Affectation

Mutex (Lock)

- Mécanisme pour implémenter l’exclusion mutuelle.
- Un seul thread ou processus peut être en possession du verrou (lock) à la fois
- Limite l’accès à une section critique

Comme les threads peuvent être bloqués et coincés en attendant qu'un thread de la section critique finisse de s'exécuter, **il est important de garder la section de code** protégée par le mutex aussi courte que possible.

Commit

- Commit **data_race.py**
- Commit **mutex.py**

Sémaphore

- Un sémaphore est un mécanisme de synchronisation qui peut être utilisé pour contrôler l'accès aux ressources partagées, comme un mutex.
- Contrairement à un mutex, un sémaphore peut permettre à plusieurs threads d'accéder à la ressource en même temps.
- Un sémaphore inclut un compteur pour suivre le nombre de fois où il a été acquis ou libéré. Tant que la valeur du compteur du sémaphore est positive, n'importe quel thread peut acquérir le sémaphore, ce qui diminue la valeur du compteur. Si le compteur atteint zéro, les threads qui essaient d'acquérir le sémaphore seront bloqués et placés dans une file d'attente pour attendre qu'il soit disponible.
- Si les valeurs possibles d'un sémaphore sont limitées à zéro ou un, zéro représentant un état verrouillé et un représentant un état déverrouillé. On parle d’un sémaphore binaire et son implémentation ressemble beaucoup à un mutex.

Commit

- Commit **semaphore.py**

Mutex vs Sémaphore

Mutex	Sémaphore
Un mutex utilise un mécanisme de verrouillage, c'est-à-dire que si un processus veut utiliser une ressource, il verrouille la ressource, l'utilise et la libère ensuite	Un sémaphore utilise un mécanisme de signalisation où les méthodes wait() et signal() sont utilisées pour montrer si un processus libère ou prend une ressource
Un mutex est un objet	Un sémaphore est une variable entière
Un mutex ne peut être déverrouillé que par le même thread qui l'a verrouillé à l'origine	Un sémaphore peut être acquis et libéré par différents threads. N'importe quel thread peut incrémenter la valeur du sémaphore en le libérant ou tenter de décrémenter sa valeur en l'acquérant.
Un objet mutex permet à plusieurs threads de processus d'accéder à une seule ressource partagée, mais un seul à la fois	Le sémaphore permet à plusieurs threads de processus d'accéder à l'instance finie de la ressource jusqu'à ce qu'elle soit disponible.

6

Monitoring

Outils préinstallés

Il y a plusieurs outils préinstallés en Python pour la mesure des performances.

- **timieit** est un module pour mesurer le temps d'execution d'un morceau de code. A chaque exécution de la fonction timeit(), il garde le résultat et en calcule la moyenne.

- **cProfile** enregistre des statistiques sur la fréquence et la durée des appels des fonctions.

- **trace** suit l'exécution du code ligne par ligne en enregistrant les appels de fonctions, les retours et exceptions. On s'en sert pour identifier les zones à optimiser.

Fonctions de timer

- Les fonctions de timer sont utilisées pour mesurer le temps d'exécution des portions de code ou des fonctions spécifiques dans une application.

- **time.perf_counter()** : fournit le temps d'exécution le plus précis disponible sur la plate-forme. Elle est généralement utilisée pour mesurer de petites durées avec une haute précision.

- **time.process_time()** : renvoie le temps de traitement CPU utilisé par le processus. Elle exclut le temps passé à attendre des ressources externes, telles que les entrées/sorties.

- **timeit** : fournit une méthode robuste pour mesurer le temps d'exécution du code. Il exécute le code plusieurs fois et calcule la moyenne des temps d'exécution pour obtenir une mesure plus précise.

Commit

- Commit **monitoring.py**

Outils et plateformes externes

- **Datadog** : Offre une visibilité en temps réel sur les applications. Datadog propose des intégrations de nombreux services et outils, des tableaux de bord et des alertes pour faciliter la détection et la résolution des problèmes de performance.

- **Dynatrace** : Dynatrace utilise l'intelligence artificielle pour identifier et résoudre les problèmes de performance. Il fournit également des analyses détaillées de l'utilisation des ressources, des transactions et des appels de service. OneAgent de Dynatrace peut être utilisé pour surveiller les applications Python sans modification de code.

- **ManageEngine Applications Manager** : Propose des intégrations avec de nombreux services et outils, y compris des applications Python.

- **Sentry :** Sentry se concentre principalement sur la surveillance des erreurs et des exceptions mais offre également des fonctionnalités de monitoring des performances, ce qui en fait un outil de monitoring complet.

7

Freezing et packaging

Freezing

- Le freezing transforme le code Python en un exécutable indépendant, différent pour chaque OS.
- Les utilisateurs n'ont pas besoin d'installer Python ou les dépendances requises pour exécuter l'application.
- Le freezing inclut généralement l'interpréteur Python et les bibliothèques nécessaires, ce qui rend l'application plus facile à déployer et à distribuer.
- Cependant, cela augmente également la taille du fichier final. Les outils courants pour le freezing des applications Python incluent PyInstaller, cx_Freeze et py2exe.

Packaging

- Le packaging organise et structure le code Python en un format distribuable.
- Il prend la forme d'un paquet ou d'une archive.
- Ces paquets sont ensuite distribués via des gestionnaires de paquets tels que pip et peuvent être installés sur d'autres systèmes qui disposent de Python.
- Les utilisateurs doivent avoir Python et les dépendances appropriées installées sur leur système pour exécuter le code.
- Le packaging est couramment utilisé pour distribuer des bibliothèques et des modules réutilisables.
- setuptools, distutils et poetry sont des outils pour créer des packages.

Avantages du freezing

- **Facilité de distribution** : Plus faciles à distribuer aux utilisateurs finaux, car pas besoin de Python ni de dépendances, juste un exécutable.
- **Compatibilité** : Moins de problèmes de compatibilité car l'exécutable inclut l'interpréteur Python et les bibliothèques nécessaires.
- **Simplicité pour les utilisateurs** Plus simples pour les utilisateurs finaux.
- **Protection du code source** : Accès plus difficile au code source de l'application.

Avantages du packaging

- **Réutilisabilité** : Les paquets Python peuvent être facilement partagés et réutilisés par d'autres développeurs, ce qui favorise la collaboration et la réutilisation du code.

- **Mise à jour facile** : Les paquets peuvent être facilement mis à jour en téléchargeant et en installant les nouvelles versions, sans avoir besoin de redéployer l'ensemble de l'application.

- **Portabilité** : Les paquets Python sont généralement plus portables que les exécutables freezés car ils peuvent être exécutés sur différentes plateformes et configurations, tant que l'interpréteur Python et les dépendances appropriées sont installés.

Outils de freezing

- PyInstaller est un outil populaire et polyvalent pour freeze des applications Python pour Windows, macOS et Linux. PyInstaller prend en charge Python 3.6 à 3.9.

- Par exemple :

  pyinstaller --onefile my_app.py

- Ensuite, vous trouverez l'exécutable dans le répertoire dist.

Packaging : choisir un nom

- Le nom de votre paquet doit être unique pour éviter les conflits avec d'autres paquets existants.
- Il doit être clair et descriptif, reflétant le but ou les fonctionnalités du paquet.
- Les noms de paquets doivent être en minuscules, avec des mots séparés par des tirets bas (_) ou des tirets (-).
- Évitez les caractères spéciaux ou les espaces.

Packaging : structuration 1/2

- Organisez le code en modules et sous-modules. Chaque module doit contenir des fonctionnalités ou des classes liées. Cela facilite la maintenance, la compréhension et la réutilisation de votre code.

- Inclure un fichier __init__.py dans chaque répertoire de du paquet. Ce fichier marqué le répertoire comme un paquet Python et peut être utilisé pour exécuter du code d'initialisation ou importer des objets depuis les modules du paquet.

- Ajouter un fichier README.md à la racine du paquet.

Packaging : structuration 2/2

- Incluez un fichier de licence pour définir les conditions d'utilisation de votre paquet. Par exemple licence MIT, BSD, GPL etc.

- Ajoutez un fichier requirements.txt à la racine pour lister les dépendances de votre projet. Ce fichier est utilisé par pip pour installer les dépendances requises.

- Créez un fichier setup.py pour décrire le paquet et ses métadonnées. Ce fichier est utilisé par les outils de packaging tels que setuptools pour empaqueter votre projet en un format distribuable.

Publication sur PyPi

- PyPI (Python Package Index) vous permet de partager votre code avec d'autres développeurs et utilisateurs Python.
- **Prérequis** : setuptools, wheel et twine.
- Votre projet doit contenir un fichier **setup.py**.
- Ensuite vous pouvez créer la distribution.
- Pour finir, inscrivez-vous sur **Pypi** et publier package à l'aide de twine.

```
twine upload dist/*
```

Commit

- Commit **pypi**

Package complet

- Un package complet est bien structuré et contient toutes les informations et fichiers nécessaires pour que les utilisateurs puissent l'installer et l'utiliser facilement.

- Utilisez un fichier **.gitignore** pour exclure les fichiers et dossiers inutiles du contrôle de version. Vous pouvez également utiliser un fichier **MANIFEST.in** pour inclure ou exclure des fichiers spécifiques de la distribution de votre package.

- Vous devez lister les dépendances dans le fichier **setup.py** sous **install_requires**.

- Description meta du package dans **setup.py**.

- Incluez une documentation et les fichiers de licence.

Commit

- Commit **package-complet**

Scripting de package 1/2

Vous pouvez utiliser des scripts pour automatiser certaines tâches, par exemple :

- **Installer les dépendances** : Créez un script qui installe toutes les dépendances de votre package.
- **Exécuter des tests** : Un script pour exécuter tous les tests de votre package en une seule commande simplifie le processus de vérification de la qualité du code et facilite l'intégration continue.
- **Générer la documentation** : Si vous utilisez des outils de génération de documentation, vous pouvez créer un script pour générer automatiquement la documentation.

Scripting de package 2/2

- **Créer de distributions** : Vous pouvez automatiser la création de distributions de votre package.
- **Publier sur PyPI** : Automatisez la publication de nouvelles versions de votre package sur PyPI avec un script qui crée une distribution et l'envoie à l'aide de twine.
- **Nettoyer le projet** : Créez un script pour supprimer les fichiers temporaires, les caches et les distributions précédentes de votre projet.

Assemblage final

- Commit **scripts**

8

Les Frameworks spécialisés

XML

- **Standards** : DOM (Document Object Model), SAX (Simple API for XML) et StAX (Streaming API for XML) sont des normes pour lire et écrire des documents XML.

- **Librairies en Python** : xml.etree.ElementTree, xml.dom.minidom et xml.sax font partie de la bibliothèque standard de Python.

- **Modules externe**s : untangle, xmltodict, lxml et Beautiful Soup (BS) sont des bibliothèques tierces pour travailler avec XML.

- **Conversion et Binding** : XPath est un langage de requête pour les documents XML, et la génération de modèles objet permet de convertir des documents XML en objets Python.

- **Sécurisation des parsers** : techniques pour éviter les attaques et les vulnérabilités liées au traitement des documents XML.

IA et ML 1/2

- **TensorFlow** : une bibliothèque développée par Google pour la création et l'entraînement de modèles de machine learning.
- **Seaborn** : une bibliothèque de visualisation de données basée sur Matplotlib, spécialement conçue pour les statistiques.
- **Numpy** : une bibliothèque pour le calcul numérique et la manipulation de tableaux multidimensionnels.
- **Pandas** : une bibliothèque pour la manipulation et l'analyse de données structurées.

IA et ML 2/2

- **Keras** : une API de haut niveau pour la création et l'entraînement de modèles de machine learning, basée sur TensorFlow ou Theano.
- **Theano** : une bibliothèque de calcul numérique pour travailler avec des expressions mathématiques impliquant des tableaux.
- **PyTorch** : une bibliothèque développée par Facebook pour l'apprentissage automatique et le calcul tensoriel.
- **Scikit-learn** : une bibliothèque pour l'apprentissage automatique, incluant divers algorithmes de classification, régression et clustering.

Sciences

- **Numpy, Pandas** : déjà mentionnés précédemment.
- **SciPy** : une bibliothèque pour les calculs scientifiques et techniques, incluant des modules pour l'optimisation, l'algèbre linéaire, l'intégration et la statistique.
- **Matplotlib** : une bibliothèque pour la création de visualisations statiques, animées et interactives en Python.

Web

- **Django** : un framework web complet pour la création d'applications web.
- **Flask** : un micro-framework web pour la création d'applications web légères et modulaires.
- **Bottle** : un micro-framework web similaire à Flask, mais encore plus léger.
- **CherryPy** : un framework web minimaliste pour la création d'applications web.
- **Falcon** : un framework web léger pour la création d'API RESTful.

9

XML ET LE WEB

XML

- Extensible Markup Language
- Le XML est très similaire au HTML, mais avec quelques modifications pour l'aider à représenter des types de données plus génériques que le simple contenu Web.
  Les règles de structure du XML sont beaucoup plus strictes
- Riche et expressif mais plus complexe que le JSON

Règle de base du formatage XML

- Les fichiers XML doivent toujours avoir une seule balise root qui contient toutes les autres balises;
- Les fichiers XML peuvent contenir une déclaration de document facultative;
- Les balises vides, c'est-à-dire les balises qui ne contiennent aucune donnée, doivent être fermées par une barre oblique à l'intérieur du nom de la balise: <tag/>
- Les balises XML qui possèdent des attributs doivent se voir attribuer des valeurs et ces dernières doivent être placées entre guillemets.
- Les balises doivent être correctement imbriquées les unes dans les autres
- La spécification XML réserve les attributs et les noms de balises qui commencent par les caractères xml
- Nota: Le XML n’impose aucun type de données

XML : exemple

```
<?xml version="1.0"?>
<Catalog>
   <Book id="bk101">
      <Author>Garghentini, Davide</Author>
      <Title>XML Developer's Guide</Title>
      <Genre>Computer</Genre>
      <Price>44.95</Price>
      <PublishDate>2000-10-01</PublishDate>
      <Description>An in-depth look at creating applications
      with XML.</Description>
   </Book>
   <Book id="bk102">
      <Author>Garcia, Debra</Author>
      <Title>Midnight Rain</Title>
      <Genre>Fantasy</Genre>
      <Price>5.95</Price>
      <PublishDate>2000-12-16</PublishDate>
      <Description>A former architect battles corporate zombies,
      an evil sorceress, and her own childhood to become queen
      of the world.</Description>
   </Book>
</Catalog>
```

JSON

- Javascript Object Notation
- Il s'agit essentiellement d'un moyen de prendre des objets arbitraires contenant des données et de représenter ces données dans un format textuel concis.
- JSON est à l'origine dérivé de JavaScript, mais il est désormais pris en charge par un large éventail de langages de programmation
- Compact et léger, facile à écrire, lire et traiter.

Règle de base du formatage JSON 1/2

- Il ne doit exister qu'un seul élément père par document contenant tous les autres : un élément racine.
- Tout fichier JSON bien formé doit être soit un objet (commençant par "{" et se terminant par "}"), soit un tableau (commençant par "[" et terminant par "]"). Cependant ils peuvent être vides, ainsi "[]" et "{}" sont des JSON valides.
- Les séparateurs utilisés entre deux paires/valeurs sont des virgules.
- Un objet JSON peut contenir d'autres objets JSON.

Règle de base du formatage JSON 2/2

- Il existe deux types d'éléments :
    - Des couples de type "nom": valeur, comme l'on peut en trouver dans les tableaux associatifs.
    - Des listes de valeurs, comme les tableaux utilisés en programmation.

JSON : valeurs possibles

- Primitifs : nombre, booléen, chaîne de caractères, null.
- Tableaux : Liste de valeurs (tableaux et objets aussi autorisés) entrées entre crochets, séparées par des virgules.
- Objets : Listes de couples "nom": valeur (tableaux et objets aussi autorisés) entrés entre accolades, séparés par des virgules.

JSON : exemple

```
{
  "squadName": "Super hero squad",
  "homeTown": "Metro City",
  "formed": 2016,
  "secretBase": "Super tower",
  "active": true,
  "members": [
    {
      "name": "Molecule Man",
      "age": 29,
      "secretIdentity": "Dan Jukes",
     "powers": ["Radiation resistance", "Turning tiny", "Radiation blast"]
    },
}
```

Module Python dans le web

Urllib	Contient plusieurs sous-modules liés au travail avec des données Internet. Tels que l'émission de requêtes et la réception de réponses d'URLs. Il gère les erreurs et les exceptions au cours de ce processus et analyse les informations relatives aux URL
Http	contient du code pour travailler avec les cookies, les serveurs et d'autres protocoles HTTP de bas niveau.
Json	Permet de convertir des données Python en JSON et des données JSON en types de données Python natifs
XML	Python dispose de plusieurs modules différents pour travailler avec le XML. On peut travailler à l'aide des API SAX et DOM, ainsi que de l'API ElementTree, qui offre un moyen plus pratique de travailler avec des données XML

Urllib

- Le module urllib est relativement simple et facile à utiliser, mais il présente plusieurs inconvénients par rapport aux autres bibliothèques disponibles.
- Urllib ne supporte qu'un sous-ensemble de l'ensemble des opérations HTTP, telles que get et post à moins que vous ne les implémentiez vous-même.
- Urllib ne décode pas automatiquement les données retournées pour vous
- Il n'y a pas de fonctionnalités intégrées à urllib pour travailler avec certaines constructions de données web communes comme les cookies, l'authentification ou les sessions. Il faudrait importer d'autres modules

Urllib.request	gère l'ouverture et la lecture des url
Urllib.error	qui définit les classes d'exceptions pour les erreurs soulevées par le module request
Urllib.parse	pour analyser les structures url
Urllib.robotparser	pour travailler avec les fichiers robots.txt

Commit

- Commit **urllib.py**

Requests

- La bibliothèque Requests simplifie et améliore considérablement les modules standards de Python.
- Requests fournit une API très simple pour travailler avec diverses opérations HTTP. Chaque opération HTTP correspond en fait à un nom de méthode dans la bibliothèque Requests
- Requests rend facile l'envoie des données supplémentaires au serveur, comme des paramètres ou des informations d'en-tête
- Requests decode automatiquement les informations renvoyées;
- Requests decode du JSON et le convertit en objets Python directement;
- Requests détecte et gère également les erreurs et les redirections
- Requests est fournie avec un ensemble de codes de réponse HTTP intégrés
- Requests prend également en charge des fonctionnalités plus avancées comme l'authentification, les objets de session et même le téléchargement de données en continu

Commit

- Commit **requests.py**

Comment manipuler du JSON

- Le module JSON fait partie de la bibliothèque standard Python intégrée et contient des méthodes pour travailler avec le format.
- Il existe deux méthodes pour convertir le texte en JSON

Load	*Obj = load(file)*	charge JSON directement depuis un fichier
Loads	*Obj = loads(string)*	Travaille directement sur une chaîne de texte

Dump	*Obj = dump(file)*	Ecrit le JSON dans un fichier
Dumps	*Obj = dumps(string)*	Ecrit le JSON dans une chaine de caractère

Conversion

JSON Data	Python Object
object	dict
array	list
string	str
Integer number	int
Floating point number	float
true, false	True, False
null	None

Python Object	JSON Representation
dict	object
list, tuple	array
str	string
int, long, float, Enums	number
True	true
False	false
None	null

Commit

- Commit **json_req.py**

Parser XML

- Les parsers XML fonctionnent essentiellement de deux manières. La première méthode est connue sous le nom de Simple API for XML, souvent appelée SAX. La deuxième méthode est appelée le Document Object Model ou simplement DOM.

- Un analyseur syntaxique qui utilise la méthode SAX fonctionne comme une chaîne de montage. Il lit les données XML un caractère à la fois jusqu'à ce que le fichier entier ait été lu. Pendant la lecture des données XML, l'analyseur génère divers événements qui correspondent au contenu du XML. Votre application Python définit donc une classe qui traitera ces événements au fur et à mesure de leur arrivée

Avantages / Inconvénients XML

- Avantages:
    - Econome en mémoire puisque le document entier n'a pas besoin d'être chargé en mémoire;
    - Rapides, puisque votre application ne répond qu'aux événements qui l'intéressent réellement
    - Faciles à mettre en œuvre
- Désavantages:
    - Aucun moyen d'accéder au contenu du XML dans n'importe quel ordre
    - Ne conserve pas d'informations contextuelles.
    - Ne peut pas modifier les données XML au cours de leur traitement

Python SAX API

Import xml.sax	L'API SAX en Python, est implémentée dans le module xml.sax
Xml.sax.parse(file, handler)	Peut être utilisé si le XML provient d'un fichier
Xml.sax.parseString(string, handler)	Peut être utilisé si le XML provient d'une variable de type chaine des caractères
Class xml.sax.Contenthandler	Classe prédéfinie qui possède des fonctions permettant de gérer le début et la fin du document, le début et la fin des balises ainsi que les données textuelles. En créant une sous classe héritant de cette classe, on peut remplacer simplement ces fonctions pour gérer chaque type de contenu et créer sa propre logique de traitement du contenu.

Commit

- Commit **sax_parse.py**

Python DOM API

Import xml.dom.minidom	bibliothèque standard Python fournit une implémentation de l'API DOM
Xml.dom.minidom.parseString(string, handler)	Peut être utilisé pour créer une structure arborescente de documents à partir du texte XML source. Pour ensuite être en mésure d'utiliser les éléments repris ci-dessous
getElementById ou *getElementByTagName*	Récupérer des éléments par ID ou par leur tagName
getAttribute ou encore *setAttribute*	Récupérer ou définir les valeurs des attributs
createElement ou createTextNode ou appendChild	Créer ou insérer un nouveau contenu

Commit

- Commit **dom_parse.py**

ElementTree API

- Element Tree API se concentre sur une manière plus simple et plus efficace de travailler avec XML et cela conduit à quelques différences importantes par rapport à DOM.
- Les éléments sont traités comme des listes
- Les attributs sont traités comme des dictionnaires
- ElementTree API fournit les fonctions *defined* et *findAll* qui rendent la recherche de contenu dans le XML très simple.

Query Expression	**Description**
Tagname	Trouver les éléments enfants immédiats avec ce nom
//tagname	Trouver l'ensemble des éléments dans le document en entier
Tagname[attr]	Rechercher les balises qui ont des attributs spécifiques
Tagname[attr = val]	Rechercher les balises qui ont des attributs avec des valeurs spécifiques

Commit

- Commit **lxml_.py**

10

Sérialisation

Définition

- La sérialisation est un besoin fréquent depuis l'avènement d'internet Lorsqu'un service doit transmettre des données à un autre, l'opération est simple lorsqu'il s'agit de transmettre un nombre réel, un entier, une chaîne de caractère. C'est un peu plus compliqué lorsque les données à transmettre sont constituées de listes et de dictionnaires. Elle est complexe lorsque ce sont des instances de classes définies par le programme lui-même.

- Le langage de programmation Python vous permet de travailler avec des structures de données de plus haut niveau telles que des listes, des dicts, des ensembles, etc.
- les services s'échangent uniquement des octets (ou byte en anglais). Il faut donc convertir des informations complexes en une séquence d'octets puis effectuer la conversion inverse. Ces opérations sont regroupées sous le terme de sérialisation.

- On rencontre le besoin de la sérialisation dans une application lors du passage de données entre programmes, par exemple lors d'un appel HTTP, ou lors du stockage de données sur disque ou dans une base de données.

- Il existe de nombreux formats de sérialisation. JSON, YAML, XML, TOML, HDF5, pickle...

Choisir le bon format

- Il existe de nombreux formats de sérialisation, des plus connus comme JSON et XML aux moins connus, comme Cap'n proto.

- Les paramètres à prendre en compte lorsque vous choisissez un format de sérialisation sont:
  - Le niveau de maturité (Utilisation, nombre des langages qui le supporte)
  - Type supporté par le format (Horodatage, Dictionnaire…)
  - Pris en compte d'un Schéma ou d'une structure
  - Performance
  - Sécurité (Certains formats de sérialisation sont plus sûrs que d'autres, mais aucun format de sérialisation n'est totalement sûr)

- Le format de sérialisation choisie doit l'être aussi sur base de certaines règles en amont:
  - Quels sont les objets à transmettre, Quelles sont leurs propriétés et quelles sont les limites des données ?
  - Quelles sont les spécificités du format que je m'apprête à choisir ?

Format de sérialisation

Format	Description
Pickle	Pickle est un format réservé à Python. Il peut sérialiser presque tous les types Python, y compris vos classes personnalisées. Il est idéal pour la communication entre deux programmes Python
JSON	Le format de serialisation le plus populaire. Il est basé sur le texte et supporte un ensemble limité de types. JSON n'intègre pas les commentaires.
YAML et TOML	Ce sont deux formats textuels très populaires pour la configuration. Ils prennent en charge une plus grande variété de types que JSON et comportent des commentaires. (pip install toml, pyYAML)
CSV & XML	Ce sont des formats anciens et établis. Le principal avantage du format CSV est que vous pouvez l'importer et l'exporter depuis Excel. Il n'y a pas de spécification de schéma pour CSV, et tout ce que vous lisez ou écrivez dans CSV doit être une chaîne. En XML, tout est également une chaîne de caractères. Il existe des schémas externes où vous pouvez définir les types.
Protocole Buffer	C'est un format binaire très efficace principalement utilisé pour la communication interne entre les services. vous écrivez un fichier .proto dans lequel vous définissez vos types de données appelés messages, puis vous utilisez le compilateur protoc pour générer du code de sérialisation pour différents langages

11

Django

Web : Common Gateway Interface CGI

- Au début d'internet, **the Common Gateway Interface (CGI)** a été conçu pour que les clients fassent fonctionner des programmes externes (de serveur web) et retourne le résultat.

- CGI a également géré les arguments en entrée pour le programme externe. Par contre, le programme reprend de zéro pour chaque accès client, ce qui fait que le programme ne scale pas très bien car tout programme à un temps de démarrage, aussi petit soit-il.

- Pour éviter ce temps de démarrage, on a fusionné le langage interpréteur avec le serveur web. PHP a le sien dans Apache avec **mod_php**, Perl dans **mod_perl** et Python dans **mod_python**.

Web : Web Server Gateway Interface WSGI

- Le développement web en Python a fait un bond en avant avec la définition d'un Web Server Gateway Interface (WSGI), un API universel entre les applications web Python et les serveurs webs. Tout framework web en Python utilise le WSGI. Vous n'avez pas besoin de connaître son fonctionnement en détail, mais c'est important de savoir qu'il existe et quel est son utilité !

Web : Frameworks

- Les serveurs Web gèrent l'HTTP et le WSGI mais vous utilisez un framework pour vraiment écrire du code Python qui va alimenter votre site.

- Si vous voulez faire un site web en Python, il existe plusieurs frameworks, les plus connus sont **Django** et **Flask**.

- Nous utiliserons **Django** dans cette formation.

Frameworks web : Fonctionnalités

- Tous ces frameworks gèrent les fonctionnalités suivantes :
  Routes, interprète les URLS et trouve le code correspondant
  Templates, fusionne les données back-end dans une page HTML
  Authentification, gère les noms d'utilisateurs, mots de passe
  Autorisation, attribut des permissions
  Sessions, maintient un stockage de données temporaires lors de la visite d'un utilisateur sur le site

Installation de Django

- Installer Django

```
python -m pip install Django
```

- Vérifier la version installée

```
>>> import django
>>> print(django.get_version())
4.1
```

- On peut commencer !

Création d'un projet

```
python -m django startproject mysite
```

- Vous devez retrouver l'arborescence ci-dessous

```
mysite/
    manage.py
    mysite/
        __init__.py
        settings.py
        urls.py
        asgi.py

        wsgi.py
```

Arborescence projet

- **manage.py :** un utilitaire pour interagir avec Django.

- **mysite/** : le répertoire de configuration du projet, son nom est le nom de votre projet.
- **mysite/__init__.py :** un fichier vide qui indique à Python que ce répertoire doit être considéré comme un module Python.
- **mysite/settings.py :** fichier de configuration du projet.
- **mysite/urls.py :** la déclaration de toutes les URL du projet.
- **mysite/asgi.py :** un point d'entrée pour les serveurs web utilisant ASGI.
- **mysite/wsgi.py :** un point d'entrée pour les serveurs web utilisant WSGI.

Serveur de développement

```
python manage.py runserver
```

- Naviguer dans le répertoire nouvellement créé
- Vous pouvez en choisir d'autres en le spécifiant.

```
python manage.py runserver 0.0.0.0:8888
```

Commit

- Commit **django**

12

TP Final

TP Final : Application complète

- Créer une application pour gérer des utilisateurs et leurs messages.
- Vous devez utiliser des context managers pour gérer les ressources, des métaclasses pour contrôler la création d'instances d'utilisateurs, des fonctions internes et des closures pour la validation des messages.
- Voir l’énoncé détaillé sur Slack.

Formation Python

Perfectionnement

Votre formateur : Quentin BIDOLET

Nous contacter :
Christophe Guéroult
c.gueroult@2aiconcept.com
+33 6 56 85 84 33