Master 2 TNAH, Ecole nationale des chartes
29 octobre 2025

Marion Charpier

Introduction au langage de programmation Python

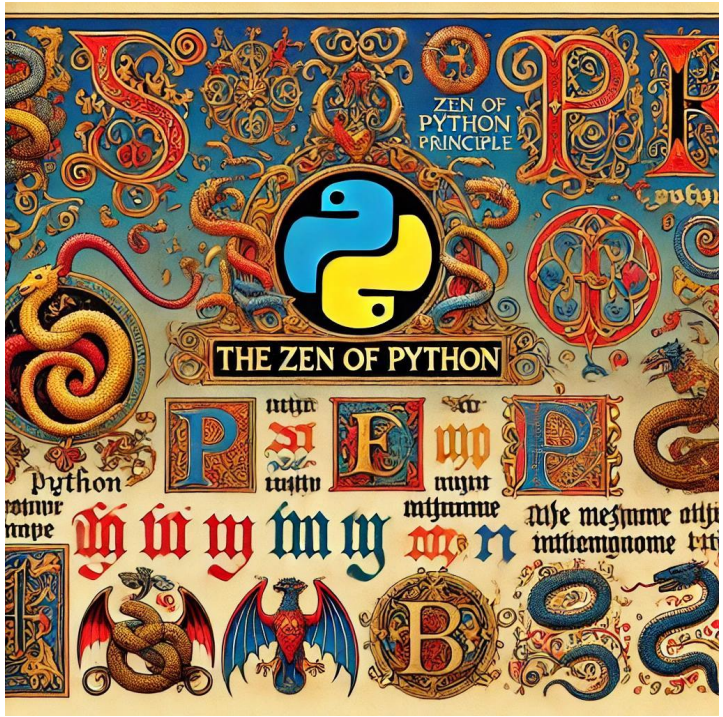
Séance 4 - Programmation orientée objet 2

Programme du cours

Séance	Date	Sujet
Séance 1	1er octobre 2025	Introduction à Python
Séance 2	14 octobre 2025	Boucles, fonctions et modules
Séance 3	28 octobre 2025	Programmation orientée objet 1
Séance 4	29 octobre 2025	Programmation orientée objet 2
Séance 5	4 novembre 2025	Manipulation de fichiers json et IIIF
Séance 6	10 novembre 2025	Manipulation de fichiers json et IIIF 2
Séance 7	18 novembre 2025	Introduction au prompt et création de scripts de gestions de données

Plan de la séance 6

1) **Révisions**
   a) Structure des manifestes
   b) Bibliothèque Json
   c) La bibliothèque Path
   d) Les décorateurs
2) **Télécharger des données avec `requests`**
   a) Accéder aux données en ligne
   b) Gérer l'accessibilité : `get` et `status_code`
   c) Téléchargement
3) **La bibliothèque `PIL`**
   a) `PIL` et `Image` : manipuler des images
   b) Méthodes principales d'Image



Structure des manifestes IIIF

Définition d'un manifeste

Qu'est-ce qu'un manifeste ?

- Un document structuré au format JSON.
- Décrit une ressource numérique complexe.
- Utilisé pour représenter des collections d'images, de textes, de vidéos, ou d'autres médias.

Utilisation dans l'histoire numérique

Avantages pour l'accès et la recherche

- Facilite l'accès aux ressources historiques à travers des plateformes et des outils standardisés.
- Favorise l'interopérabilité des ressources en ligne entre institutions.

Impact sur la recherche et la diffusion

- Améliore la consultation et l'étude de collections numériques.
- Encourage les collaborations internationales en histoire numérique.

Éléments clés

- **@id** : Identifiant unique de la ressource.
- **@type** : Type de ressource (par exemple, sc:Manifest pour un manifeste IIIF).
- **Métadonnées** :
  - Informations supplémentaires pour décrire la ressource.
  - Exemples : titre, description, créateur, format, droits, etc.

Cette structure des manifestes permet d'organiser et de partager des ressources complexes de manière standardisée et interopérable dans le cadre de projets patrimoniaux et éducatifs en ligne.

Les canvas

Qu'est-ce qu'un canvas ?

- Un "canvas" est une unité de contenu individuelle.
- Représente des éléments comme une page, une image individuelle ou toute autre ressource spécifique.

Rôle du canvas dans IIIF

- Sert de base pour afficher et organiser des ressources multimédia sur une plateforme IIIF.
- Chaque canvas agit comme un espace virtuel pour y placer du contenu multimédia via des annotations.

Grâce aux canvas, IIIF permet une gestion structurée et une navigation intuitive au sein des collections numériques, facilitant la consultation et l'exploration des documents historiques et culturels.

Structure d'un Canvas

Propriétés clés :

- **Annotations** : Descriptions et liens vers des ressources associées (images, textes, etc.).
- **Dimensions** : Largeur et hauteur du canvas pour garantir un rendu précis de la ressource.
- **Liens vers les ressources** : Accès aux ressources multimédia (images, vidéos, textes) liées au canvas.

Rôle dans la Navigation

Organisation de la collection :

- Les canvas structurent la collection en organisant chaque ressource individuelle (page, image) de manière cohérente.

Navigation à travers des documents complexes :

- Enchaîner les canvas permet de parcourir facilement un document multi-pages (comme un manuscrit ou un livre).
- Facilite l'expérience utilisateur en permettant une navigation fluide entre les unités de contenu.

La bibliothèque `json`

Rappel sur la bibliothèque

- **Présentation de JSON**
  - JavaScript Object Notation.
  - Format léger d'échange de données.
- **Importance en Python**
  - Bibliothèque intégrée json.
  - Manipulation facile des données JSON.
- **Cas d'utilisation**
  - Lecture de configurations.
  - Échange de données avec des API.
  - Stockage de données structurées.

Fonctions principales

`json.load(fichier)`

- Charge un objet JSON depuis un fichier.

`json.loads(chaîne)`

- Charge un objet JSON depuis une chaîne de caractères.

`json.dump(objet, fichier)`

- Écrit un objet JSON dans un fichier.

`json.dumps(objet)`

- Convertit un objet Python en chaîne JSON.

Options utiles

- `indent` : pour une sortie formatée
- `sort_keys` : pour trier les clés

La bibliothèque `Path`

Pourquoi utiliser `Path` ?

`Path` est la façon moderne de travailler avec les fichiers et dossiers en Python.
Au lieu de manipuler des **chaînes de caractères**, on manipule des **objets "chemin"** qui savent déjà où ils se trouvent et ce qu'ils représentent.

Cela rend le code :

- **plus lisible** (on comprend mieux ce qu'on fait),
- **plus sûr** (moins d'erreurs de concaténation de chemins),
- et **portable** (il fonctionne de la même manière sur Windows, macOS et Linux).

Principales Fonctions de `pathlib`

- `p.name` → nom du fichier
- `p.stem` → nom sans extension
- `p.suffix` → extension
- `p.parent` → dossier parent

Vérifications et opérations de base

- `p.exists()` → vérifie si le chemin existe
- `p.is_file()` / `p.is_dir()` → teste si c'est un fichier ou un dossier
- `p.resolve()` → renvoie le chemin absolu

Combinaison de chemins :

```
p = Path("data") / "images" / "photo.png"
```

Les décorateurs

Qu'est-ce qu'un décorateur ?

- Un **décorateur** est une fonction qui prend en entrée une autre fonction et retourne une nouvelle fonction avec un comportement modifié ou étendu.
- Il permet de modifier dynamiquement le comportement d'une fonction **sans changer son code source**.

A NOTER

- Lorsque `dire_bonjour()` est appelée, c'est en réalité `nouvelle_fonction()` qui est exécutée, avec le code supplémentaire.
- Le **code supplémentaire** autour de la fonction originale est exécuté en premier, puis la fonction d'origine est appelée.

Ajouter des fonctionnalités sans modifier le code source

- Permet d'enrichir une fonction existante en y ajoutant des fonctionnalités supplémentaires.
- Idéal pour **respecter le principe Open/Closed** : les entités logicielles doivent être ouvertes à l'extension mais fermées à la modification.

Séparation des préoccupations

- Permettent de **séparer le code métier** du code technique (par exemple, la journalisation, la gestion des exceptions, la vérification des permissions).
- Améliorent la **lisibilité** et la **clarté** du code.

Réutilisation du code

- Les décorateurs favorisent la **réutilisation** en permettant d'appliquer la même fonctionnalité à plusieurs fonctions sans dupliquer le code.
- Facilitent la **maintenance** et l'**évolution** du code en centralisant les modifications.

Comment fonctionnent les décorateurs ?

Fonctionnement

Le décorateur enveloppe la fonction originale :

- Il peut ajouter du **code avant et après** l'exécution de la fonction cible.

Définition du décorateur :

- Un décorateur est une **fonction** qui accepte une fonction en argument et retourne une nouvelle fonction avec un comportement modifié.

Application du décorateur

Utilisation du symbole `@` :

- Le décorateur est appliqué à une fonction cible en utilisant `@nom_du_décorateur` juste au-dessus de la définition de la fonction.

Exécution

- Lorsque `ma_fonction()` est appelée, c'est en réalité `fonction_modifiee()` qui est exécutée, avec le code supplémentaire.
- Le **code supplémentaire** autour de la fonction originale est exécuté en premier, puis la fonction d'origine est appelée.

Exemple

```
def mon_decorateur(fonction):
     def nouvelle_fonction(*args, **kwargs):
     # Code avant
     resultat = fonction(*args, **kwargs)
     # Code après
     return resultat
     return nouvelle_fonction
```

La bibliothèque `requests`

La bibliothèque `requests` est un module Python extrêmement populaire pour effectuer des requêtes HTTP de manière simple et efficace. Elle permet de communiquer avec des APIs, télécharger des fichiers, ou récupérer des données à partir du Web.

Principales fonctions et attributs de requests

`requests.get(url)`

- Effectue une requête HTTP GET pour récupérer des données d'une URL donnée.

`response.content`

- Renvoie le contenu brut de la réponse, en bytes. Utilisé pour télécharger des fichiers binaires (images, vidéos, etc.).

`response.json()`

- Convertit automatiquement la réponse en format JSON (si la réponse est JSON), pratique pour accéder à des données structurées.

Manipulation des codes et gestion des exceptions

- `.status_code` : Permet de vérifier si la requête a réussi.

Il est important de connaître la signification des codes de retour pour comprendre les erreurs et les gérer correctement.

- **Codes de statut HTTP**
  - `200 OK`,
  - `404 Not Found`,
  - `500 Internal Server Error`, etc.
- **Gestion des erreurs**
  - Conditions basées sur le code de statut.
- **Exceptions de `requests`**
  - Utilisation de `try...except` pour capturer les erreurs.

Téléchargement de fichier avec `requests`

Télécharger une ressource

```
url = 'https://example.com/resource'
response = requests.get(url)
```

Enregistrer une ressource

```
with open('fichier.ext', 'wb') as file:
     file.write(response.content)
```

Vérification du Type de Contenu

Lorsqu'on effectue une requête HTTP pour obtenir une ressource, il est essentiel de vérifier que le type de contenu reçu correspond bien à ce qu'on attend (par exemple, une image, un document JSON, etc.). Cela se fait en vérifiant l'en-tête Content-Type dans la réponse :

Exemple : `response.headers['Content-Type']`
`Content-Type` : Cet en-tête HTTP indique le type MIME de la ressource (par exemple, `image/jpeg` pour une image JPEG, `application/json` pour un fichier JSON, etc.).
Cette vérification est particulièrement utile pour éviter d'analyser ou de traiter des fichiers de types inattendus, ce qui pourrait entraîner des erreurs.

Gestion des xceptions avec raise_for_status()

Gestion des exceptions avec raise_for_status()
response.raise_for_status()
→ Vérifie le code HTTP et **lève une erreur** (HTTPError) si la requête a échoué.

- **Erreurs client (4xx)** ou **serveur (5xx)** → exception automatique
- **Avantage :** plus besoin de tester manuellement response.status_code

Exemple

```
try:
      response = requests.get(url)
      response.raise_for_status()
except requests.exceptions.RequestException as e:
      print(f"Erreur lors du téléchargement : {e}")
```

Explication

1. **Envoi de la requête**
   `response = requests.get(url)` tente de récupérer la ressource à l'URL spécifiée.
2. **Vérification d'erreurs avec `raise_for_status()`** : Si la requête échoue (par exemple, statut 404 Not Found), `.raise_for_status()` lève une exception.
3. **Gestion des exceptions** :
   - **Bloc `try`** : En cas de succès, le code se poursuit normalement.
   - **Bloc `except`** : Si une exception se produit, elle est capturée par `requests.exceptions.RequestException` (qui couvre toutes les erreurs possibles du module requests).
   - Le message d'erreur est ensuite affiché avec `print(f"Erreur lors du téléchargement : {e}")`, permettant de diagnostiquer la cause de l'erreur.

Manipulation des fichiers en Python avec `open()`

`with open(filename, mode)` : Ouvre un fichier pour le lire, écrire, ou créer.

Modes d'ouverture courants :

Mode	Description
"r"	Lire (Erreur si le fichier n'existe pas)
"w"	Écrire (Crée ou écrase le fichier)
"a"	Ajouter à la fin (Crée le fichier si inexistant)
"r+"	Lire et écrire
"wb"	Écriture binaire (images, audio, etc.)
"w+"	Lire et écrire (Écrase le contenu)

Lire un fichier :

- `read()` : Lit tout le contenu.
- `readline()` : Lit une ligne.
- `readlines()` : Lit toutes les lignes sous forme de liste.

Écrire dans un fichier :

- `write(data)` : Écrit une chaîne de caractères ou des données binaires.
- `writelines(list)` : Écrit une liste de lignes.

Bonne pratique :

- Utiliser **`with open(...)`** pour une gestion automatique des ressources.

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

La bibliothèque PIL

Présentation de PIL / Pillow

- **PIL** : Python Imaging Library, une bibliothèque pour le traitement d'images en Python.
- **Pillow** : Un fork de PIL, activement maintenu et enrichi de nouvelles fonctionnalités, devenu la version standard utilisée aujourd'hui.

Fonctionnalités de Pillow

- **Support de nombreux formats d'images** : JPEG, PNG, BMP, GIF, TIFF, et bien d'autres.
- **Traitements avancés** :
  - **Filtres** : Appliquer des effets (flou, netteté, etc.).
  - **Transformations** : Rotation, redimensionnement, recadrage, etc.
  - **Manipulations de couleur** : Conversion en niveaux de gris, ajustement de contraste, etc.

Classe `Image` de `PIL`

- **Chargement et création d'images** : La classe `Image` permet de charger des images depuis des fichiers ou de créer de nouvelles images vierges.

```
From PIL import Image
```

Méthodes principales d'`Image`

Image.open() : Ouvre une image à partir d'un fichier

```
from PIL import Image
# Ouvrir une image existante
image = Image.open("example.jpg")
```

save() : Sauvegarde l'image dans un format spécifique

```
# Sauvegarder l'image dans un autre format (par exemple, PNG)
image.save("example_converted.png")
```

show() : Affiche l'image dans le visualiseur par défaut

```
# Afficher l'image dans le visualiseur d'images par défaut du système
image.show()
```

`resize()` : Redimensionne l'image

```
# Redimensionner l'image à une taille de 200x200 pixels
resized_image = image.resize((200, 200))
```

`rotate()` : Fait pivoter l'image selon un angle donné

```
# Faire pivoter l'image de 90 degrés
rotated_image = image.rotate(90)
```

`crop()` : Recadre l'image à partir de coordonnées spécifiées

```
# Recadrer l'image avec les coordonnées (gauche, haut, droite, bas)
cropped_image = image.crop((100, 100, 400, 400))
```

Notions à retenir

Bases et révisions

- Structure des **manifestes**
  Manipulation des données avec la **bibliothèque json**
- Gestion des chemins et fichiers avec **pathlib**

Programmation et automatisation

- Utiliser les **décorateurs** pour enrichir les fonctions
- Télécharger des données en ligne avec **requests**
- Contrôler l'accessibilité : **get()**, **status_code**, **raise_for_status()**
- Enregistrer et gérer les **fichiers téléchargés**

Images et traitement graphique

- Découverte de la **bibliothèque PIL** (Pillow)
- Utiliser l'objet **Image** pour ouvrir, afficher, modifier et sauvegarder des images
- Principales méthodes : **open()**, **show()**, **resize()**, **save()**