Master 2 TNAH, Ecole nationale des chartes
5 novembre 2024

Introduction au langage de programmation Python

—

Séance 6 - Introduction à IIIF et à la manipulation de fichiers 2

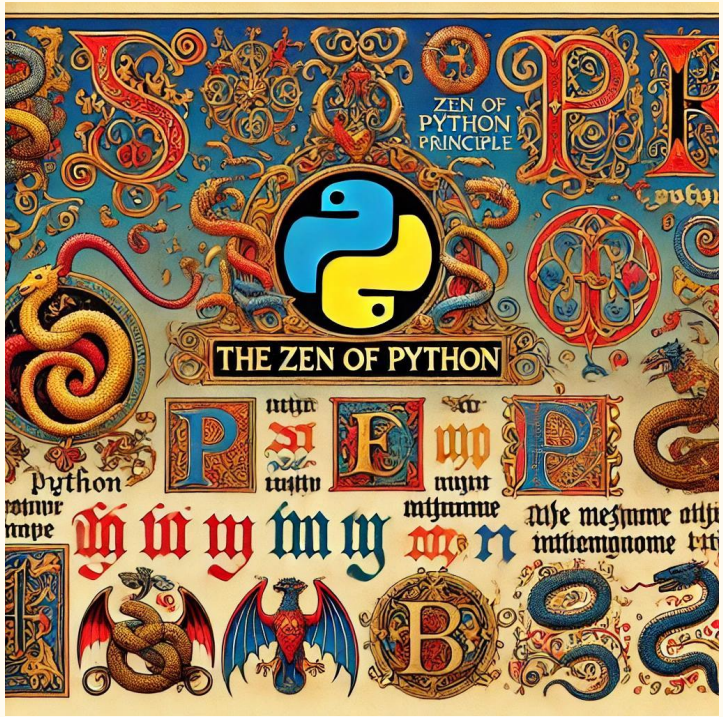
Marion Charpier

Programme du cours

Séance	Date	Sujet
Séance 1	1er octobre 2024	Introduction à Python
Séance 2	8 octobre 2024	Fonctions et Modules
Séance 3	15 octobre 2024	Programmation orientée objet 1
Séance 4	22 octobre 2024	Programmation orientée objet 2
Séance 5	29 octobre 2024	Utilisation de l'API IIIF et manipulation de données 1
Séance 6	**5 novembre 2024**	**Utilisation de l'API IIIF et manipulation de données 2**

Plan de la séance 6

1) Révisions
   a) Structure des manifestes
   b) Bibliothèque Json
   c) La bibliothèque os
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 `os`

Intérêt de `os`

- **Interaction avec le système d'exploitation**
  - Gestion des fichiers et répertoires.
  - Exécution de commandes système.
- **Portabilité du code**
  - Code compatible Windows, macOS, Linux.
- **Accès aux variables d'environnement**
  - Configuration des applications.

Création de filepath

- **Utilisation de `os.path`**
  - Chemins indépendants du système.
- **Fonctions utiles**
  - `os.path.exists()`: Vérifie si un chemin existe.
  - `os.path.dirname()`: Répertoire parent.
  - `os.path.basename()`: Nom du fichier.
- **Résolution de chemins absolus**
  - `os.path.abspath()`.

Création de dossier

- **Créer un répertoire**
  - `os.mkdir('nouveau_dossier')`.
- **Créer des répertoires imbriqués**
  - `os.makedirs('dossier1/dossier2/dossier3')`.
- **Gérer les exceptions**
  - Vérifier l'existence avec `os.path.exists()`.

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.
  - Retourne un objet `Response`, qui contient les données de réponse et les métadonnées associées.
- `response.content`
  - Renvoie le contenu brut de la réponse, en bytes. Utilisé pour télécharger des fichiers binaires (images, vidéos, etc.).
- `response.text`
  - Renvoie le contenu de la réponse sous forme de chaîne de caractères, idéal pour du texte brut comme du HTML ou du JSON.
- `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 Exceptions avec raise_for_status()

La méthode .raise_for_status() dans le module requests vérifie le statut de la réponse HTTP et déclenche une exception si le statut indique une erreur (statuts 4xx pour les erreurs côté client et 5xx pour les erreurs côté serveur).

- **Fonctionnement** : Si la requête a échoué (par exemple, avec un statut 404 ou 500), .raise_for_status() lève une exception HTTPError.
- **Avantage** : Cela permet de gérer automatiquement les erreurs HTTP sans avoir à vérifier manuellement le code de statut de chaque réponse.

Exemple

```
try:
        response = requests.get(url)
        response.raise_for_status()  # Vérifie si une erreur HTTP s'est produite
except requests.exceptions.RequestException as e:
        print(f"Erreur lors du téléchargement : {e}")
```

Explication de l'exemple

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()`

`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))

# Sauvegarder la version redimensionnée
resized_image.save("example_resized.jpg")
```

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

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

# Sauvegarder la version pivotée
rotated_image.save("example_rotated.jpg")
```

- **`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))

# Sauvegarder la version recadrée
cropped_image.save("example_cropped.jpg")
```

Conclusion

- **Maîtrise des bibliothèques essentielles**
  - `json`, `os`, `requests`, `PIL`.
- **Applications en histoire numérique**
  - Gestion et manipulation des données et des images.
- **Prochaines étapes**
  - Intégration de ces outils dans vos projets.
  - Exploration de fonctionnalités avancées.