Master 2 TNAH, Ecole nationale des chartes
29 octobre 2024

Introduction au langage de programmation Python

—

Séance 5 - Introduction à IIIF et à la manipulation de fichiers

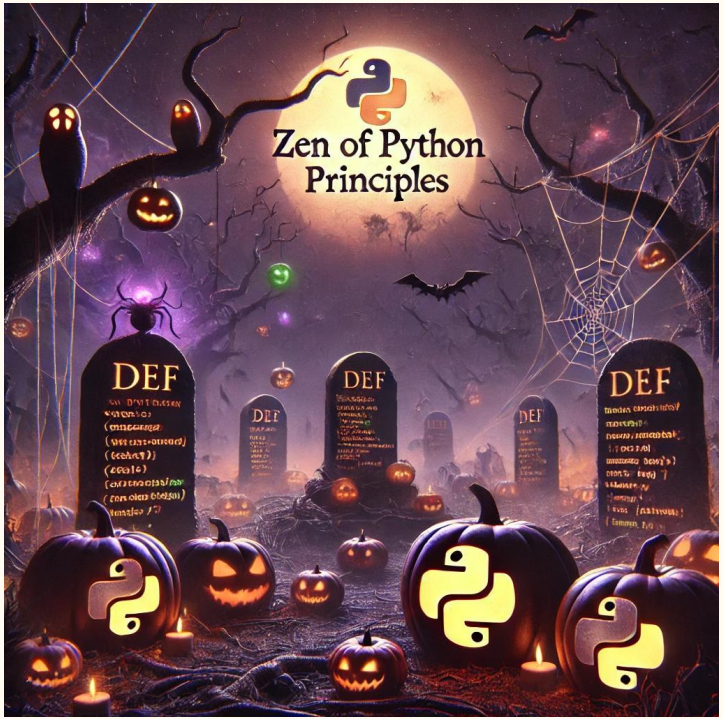
Marion Charpier

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 5

1) Révisions
   a) Héritage
   b) Décorateurs
   c) Les paramètres *args et *kwargs
2) Introduction à IIIF
   a) Structure des Manifestes IIIF
   b) La bibliothèque json
   c) Parcourir un manifest
3) Manipuler des images
   a) Gérer le téléchargement avec os
   b) Téléchargement simples d'images (PIL)
   c) Gérer les téléchargement multiples



Rappel : L'Héritage en POO

Définition

L'héritage est un mécanisme en POO qui permet à une classe (appelée **classe enfant** ou **sous-classe**) d'**hériter** des attributs et méthodes d'une autre classe (appelée **classe parente** ou **super-classe**).

Établit une **hiérarchie** entre les classes pour une meilleure organisation du code.

Utilisation de `super()`

- Appelle les méthodes de la classe parente.
- Assure une **initialisation correcte** des attributs hérités.

Structure d'une Classe Enfant avec `super()`

```
class ClasseEnfant(ClasseParente):
	def __init__(self, attP1, attP2, attE1, attE2):
	super().__init__(attP1, attP2)
	self.attE1 = attE1
	self.attE2 = attE2
```

Utilisation de `super().methode_parente()` :

- Permet d'appeler une méthode de la classe parente depuis la classe enfant :
    - **Réutiliser le comportement de la classe parente :** Profiter des fonctionnalités déjà définies pour ne pas réécrire le code existant.
    - **Étendre ou modifier le comportement :** Ajouter du code supplémentaire ou personnaliser le comportement pour répondre aux besoins spécifiques de la classe enfant.

Exemple

```
class ClassePersonne:
	# Code et méthodes de la classe parente

class JeMePresente(ClassePersonne):
	def __init__(self, nom, prenom, age):
	super().__init__(nom, prenom)
	self.age = age

	def se_presenter(self):
	super().se_presenter() # Appelle la méthode de la
classe parente

	return f"Je m'appelle {self.prenom} {self.nom} et j'ai
{self.age} ans" # Code supplémentaire spécifique à la classe
enfant
```

Les décorateurs

Qu'est-ce qu'un décorateur ?

- **Définition :**
  - 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.

Syntaxe avec `@` :

```
def mon_decorateur(fonction):
        def nouvelle_fonction():
        print("Avant l'exécution de la fonction")
        resultat = fonction()
        print("Après l'exécution de la fonction")
        return resultat
return nouvelle_fonction

@mon_decorateur
def dire_bonjour():
        print("Bonjour !")
```

Les paramètres *args et *kwargs

`*args` : Appel des arguments positionnels

- Permet de passer un nombre variable d'arguments à une fonction.
- Les arguments sont accessibles sous forme de tuple.
- Pratique quand on ne connaît pas à l'avance le nombre d'arguments ou, dans le cas des décorateurs pour être appliqué à des fonctions dont le nombre d'arguments peut varier

Exemple

```
def somme(*args):
        total = sum(args)
        print(f"La somme est : {total}")

somme(1, 2, 3, 4)  # La somme est : 10
```

****kwargs : Appel des arguments nommés**

- Permet de passer un nombre variable de paires clé-valeur à une fonction.
- Les arguments sont accessibles sous forme de dictionnaire.
- Utile pour les fonctions qui acceptent des paramètres optionnels.

```
def afficher_info(**kwargs):
        for clé, valeur in kwargs.items():
        print(f"{clé} : {valeur}")

afficher_info(nom="Charpier", prenom="Marion", ville="Paris")

# Affiche :
nom : Charpier
prenom : Marion
ville : Paris
```

Utilisation combinée de `*args` et `**kwargs`

- Possible de les utiliser ensemble dans une même fonction.
- `*args` doit être placé avant `**kwargs`.

```
def exemple(*args, **kwargs):
        print("args :", args)
        print("kwargs :", kwargs)

exemple(1, 2, 3, nom="Alice", age=25)
```

IIIF
International Image Interoperability Framework

Qu'est-ce que IIIF ?

IIIF désigne à la fois une **communauté** et un **cadre d'interopérabilité** pour diffuser, présenter et annoter des images et documents audio/vidéo sur le Web.

Un nom transparent

- **Framework** : Un *framework* est un ensemble structuré d'outils, de bibliothèques et de conventions qui fournit une base de développement facilitant la création et la structuration d'applications logicielles.
- **Interopérabilité** : la capacité de différents systèmes, logiciels ou dispositifs à travailler ensemble de manière transparente, malgré leurs différences techniques, afin de partager des informations et d'accomplir des tâches communes.

Un objectif clair et ambitieux

Décloisonner des collections numériques des institutions patrimoniales à l'échelle mondiale afin d'offrir un espace commun de recherche et de navigation.

API Présentation

L'API Présentation est à la fois **un format** d'échange et **un modèle décrivant la représentation numérique** d'un objet, sa structure interne, ses métadonnées, ses liens avec d'autres ressources.

Cette **API spécifie** également les **métadonnées techniques** nécessaires à la présentation d'un objet numérique dans une interface (par exemple un visualiseur d'images, ou tout autre environnement manipulant des images et autres médias supportés par IIIF).

Ces informations sont contenues dans un fichier appelé **Manifeste** qui sert d'enveloppe virtuelle et constitue l'unité de distribution élémentaire dans l'univers IIIF.

Ce manifeste est l'objet que vont manipuler les logiciels pour interagir avec une ressource, la visualiser, l'importer, ou la transférer vers un autre outil.

Manifeste IIIF

L'**API Présentation** est le modèle sous-jacent selon lequel est construit un Manifeste. Cette API constitue à la fois :

- un **format d'échange**, sérialisé en JSON-LD
- un **modèle de données** décrivant la représentation numérique d'un objet, qu'il soit numérisé ou nativement numérique.

Json est un format de description et de transmission de données structurées, inspiré du XML, permettant d'encapsuler des données liées à un objet:

- Soit une image ou une vidéo
- Les métadonnées associées : https://media.getty.edu/iiif/image/5a4ff989-f6a7-4bdb-816c-2dfabae437a7/info.json
- Permet également d'encapsuler des données liées d'un ensemble d'objet dont la structuration est donnée par le manifeste.

Exemple : https://gallica.bnf.fr/iiif/ark:/12148/btv1b525125689/manifest.json

Le format JSON

Qu'est-ce que JSON ?

- **JavaScript Object Notation**
- Format léger et lisible pour l'échange de données
- Utilisé largement pour les API web et les services de données
- Structure basée sur des **paires clé-valeur**

Structure d'un objet JSON

- Composé de **paires clé-valeur** ou de **listes**
- Les clés sont toujours des **chaînes de caractères**
- Les valeurs peuvent être :
  - Chaînes de caractères (`"texte"`)
  - Nombres (`123`)
  - Booléens (`true` / `false`)
  - Tableaux (`[]`)
  - Objets (`{}`)

Exemple de structure JSON

```json
{
  "titre": "Halloween",
  "annee": 1978,
  "realisateur": "John Carpenter",
  "suite": true,
  "personnages_principaux": [
      {
      "nom": "Laurie Strode",
      "acteur": "Jamie Lee Curtis"
      },
      {
      "nom": "Michael Myers",
      "acteur": "Nick Castle"
      }
  ],
  "genres": ["Horreur", "Slasher"]
}
```

Manifeste IIIF

- Si le format est le même pour tous les manifestes les métadonnées et leur description est propre à chaque institution

  Ex : https://media.getty.edu/iiif/manifest/1c76b1df-5f43-4340-bf2a-a9200d92142a

Il existe également des manifestes d'annotations, ne contenant aucune image mais servant de 'conteneur' pour les annotations (description, transcription, etc.). L'intérêt de dissocier l'annotation du manifeste contenant les objets est :

- D'alléger les données et donc le temps de chargement,
- De modifier facilement si besoin les annotations,
- De ne pas délivrer de l'information 'inutile'. Les données d'annotations ne sont pas nécessaires à tous les utilisateurs.

  Ex : https://iiif.bodleian.ox.ac.uk/iiif/annotationlist/3da059d4-e824-4082-832d-7ee2705fb9af.json

Structure des manifestes

JSON-LD est une extension de JSON qui permet d'**intégrer des données liées** (Linked Data) dans des documents JSON. Il est conçu pour rendre les données facilement compréhensibles et exploitables par les machines, **facilitant ainsi l'interopérabilité** des données sur le web.

Dans le JSON-LD, l**es clés précédées d'un `@`** sont des **mots-clés réservés** qui ont une signification particulière dans le contexte des données liées. Cela permet de les distinguer des autres clés personnalisées dans le manifeste, afin de garantir une meilleure interopérabilité des données.

Composants principaux :

- `@context` : **Spécifie le contexte JSON-LD**, qui définit les termes utilisés dans le document. Le contexte permet d'**établir des correspondances entre les termes** locaux utilisés dans le document JSON **et les concepts du web sémantique.** Cela permet de préciser les significations des clés et **d'assurer l'interopérabilité** avec d'autres données.
- `@id` : **Représente l'identifiant unique d'une ressource sur le web.** Dans un manifeste IIIF, il s'agit **souvent de l'URL** qui identifie un objet particulier (comme un manuscrit, une image, ou un autre type de ressource). Cet identifiant permet d'accéder à la ressource ou de la référencer de manière standard.
- `@type` : **Indique le type de la ressource.** Cela permet de spécifier la nature de l'objet (par exemple, sc:Manifest pour un manifeste, ou sc:Canvas pour une toile). Cela aide les applications à comprendre le type d'entité qu'elles traitent.
- `label` : **Titre ou nom de la ressource**.
- `metadata` : **Liste de métadonnées supplémentaires** (les éléments qui la compose et son exhaustivité sont propres à chaque institution/oeuvre).
- `sequence` : **Liste des éléments qui structurent l'oeuvre** (ordre des pages, des folios etc.).

La bibliothèque `json`

La bibliothèque `json` est intégrée à Python et permet de travailler avec des données au format JSON.

Fonctions principales :

- `json.load(fichier)`
  - Charge un objet JSON depuis un fichier.
  - **Exemple :**

```
with open("film.json", "r") as fichier:
data = json.load(fichier)
```

- `json.loads(chaîne)`
  - Charge un objet JSON depuis une chaîne de caractères.
  - **Exemple :**

```
json_str = '{"titre": "Halloween", "annee": 1978}'
data = json.loads(json_str, indent=4)
```

- `json.dump(objet, fichier)`
  - Écrit un objet JSON dans un fichier.
  - **Exemple :**

```
with open("film.json", "w") as fichier:
json.dump(data, fichier)
```

- `json.dumps(objet)`**
  - Convertit un objet Python en chaîne JSON.
  - **Exemple :**

```
json_str = json.dumps(data, indent=4)
```

- **Options utiles**
  - `indent` : pour une sortie formatée
    Ex. : `indent=4`
  - `sort_keys` : pour trier les clés
    Ex. : `sort_keys=True`

API Image

L'**API Image** est l'interface qui permet de délivrer des images numériques en haute résolution selon un protocole de diffusion cohérent afin de faciliter leur exploitation.

Apports de l'API Image de IIIF :

- une **syntaxe d'URL standard** pour accéder à une image et la manipuler à distance
- des **informations techniques sur l'image** pour qu'un visualiseur puisse l'afficher dans différents contextes
- des URL d'images modifiable manuellement en fonction des besoins
- des URL facilitant la mise en cache

URL IIIF

Deux modèles d'URL : une requête de l'image elle-même (pixels) et une requête d'information sur l'image (JSON)

Schéma de l'URL image :

{scheme}://{server}{/prefix}/{identifier}/{region}/{size}/{rotation}/{quality}.{format}

Ex: https://media.getty.edu/iiif/image/5a4ff989-f6a7-4bdb-816c-2dfabae437a7/full/full/0/default.jpg

Schéma de l'URL d'information :

{scheme}://{server}{/prefix}/{identifier}/info.json

Ex: https://media.getty.edu/iiif/image/5a4ff989-f6a7-4bdb-816c-2dfabae437a7/info.json

Gérer le téléchargement d'image les bibliothèques `os` et `requests`

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.status_code`
  - Permet de vérifier si la requête a réussi.
  - Exemple de codes :
    - 200 : Succès.
    - 404 : Non trouvé.
    - 500 : Erreur serveur.
- `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.

Bibliothèque `os`

La bibliothèque `os` fait partie de la bibliothèque standard de Python et permet d'interagir avec le système d'exploitation. Elle est surtout utilisée pour manipuler les fichiers, les dossiers et les chemins.

Principales Fonctions de `os`

- `os.path.join()`
  - Combine des éléments de chemin en un chemin complet compatible avec le système d'exploitation.
  - **Ex :** `file_path = os.path.join('folder', 'sub_folder', 'file')`
- `os.makedirs()`
  - Crée un dossier (et tous les dossiers parents si nécessaire). Utile pour créer un chemin de manière récursive.
  - `exist_ok=True` permet d'éviter une erreur si le dossier existe déjà.
  - **Ex :** `os.makedirs(new_folder, exist_ok=True)`
- `os.remove()`
  - Supprime un fichier spécifique.
- `os.listdir()`
  - Liste le contenu d'un dossier donné. INDISPENSABLE pour traiter des fichiers multiples dans un dossier
- `os.path.exists()`
  - Vérifie si un fichier ou un dossier existe à un chemin donné. Retourne un booléen.
- `os.getcwd()`
  - Renvoie le répertoire de travail actuel.
  - **Ex :** `print(os.getcwd())  # Répertoire actuel`

Manipulation d’image avec PIL

Qu'est-ce que PIL (Pillow) ?

- **Pillow** est une bibliothèque Python permettant de manipuler facilement des images.
- C'est une version améliorée et maintenue de l'ancienne bibliothèque **PIL (Python Imaging Library)**.
- Elle permet de **charger, afficher, modifier, et sauvegarder** des images dans divers formats (JPEG, PNG, GIF, etc.).

Principales fonctionnalités de Pillow

- **Chargement et affichage d'images**.
- **Transformation d'images** : redimensionnement, rotation, recadrage, conversion en niveaux de gris, etc.
- **Manipulation avancée** : dessin de formes, ajout de texte, filtres, etc.

Méthodes et attributs essentiels pour commencer

1. `Image.open()` : Charge une image à partir d'un fichier.
   - Utilisé pour obtenir un objet image prêt pour des manipulations.
   - **Ex :** `img = Image.open("chemin/vers/image.jpg")`
2. `.show()` : Affiche l'image dans une fenêtre.
   - Idéal pour une prévisualisation rapide.
   - **Ex :** `img.show()`
3. `.save()` : Enregistre l'image dans un fichier avec le format spécifié.
   - **Ex :** `img.save("chemin/vers/image_nouvelle.jpg", "JPEG")`
4. **Attributs de l'objet image**
   - `img.filename` : Chemin absolu de l’image.
   - `img.format` : Format de l’image (JPEG, PNG, etc.).
   - `img.size` : Dimensions (`width, height`) en pixels.