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 csv
Séance 7	18 novembre 2025	Introduction au prompt et création de scripts de gestions de données

Plan de la séance 5

1) **Révisions**
   a) Héritage
   b) 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 pathlib
   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
```

Rappel : L'Héritage en POO

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 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

```
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 key, value 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 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.

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.

API Présentation

Qu'est-ce que l'API Présentation ?
L'API Présentation est à la fois un **format d'échange** et un **modèle de description** qui définit la **représentation numérique d'un objet**, sa **structure interne**, ses **métadonnées** et ses **liens** avec d'autres ressources.

Rôle et fonction
Cette API précise les **métadonnées techniques** nécessaires à la **présentation d'un objet numérique** dans une interface :

- visualiseur d'images,
- outil d'annotation,
- ou tout autre environnement compatible avec IIIF.

Elle permet ainsi une **interprétation cohérente et normalisée** des contenus par différents logiciels.

Le Manifeste IIIF

Les informations de l'API Présentation sont rassemblées dans un **fichier appelé "Manifeste"**, qui joue le rôle d'**enveloppe virtuelle**.
Ce manifeste constitue **l'unité de distribution élémentaire** dans l'écosystème IIIF.

C'est **l'objet manipulé par les logiciels** pour :

- afficher une ressource,
- l'importer dans une autre application,
- ou la transférer vers un nouvel environnement.

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
- un **modèle de données** décrivant la représentation numérique d'un objet, qu'il soit numérisé ou nativement numérique.

Définition

Le **Manifeste IIIF** est construit selon le **modèle défini par l'API Présentation**.

Cette API constitue à la fois :

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

Le format JSON-LD

JSON (*JavaScript Object Notation*) est un **format de description et de transmission de données structurées**, inspiré du XML.

Il permet d'**encapsuler des informations liées à un objet**, telles que :

- une **image** ou une **vidéo**,
- les **métadonnées associées**, par exemple :

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

Unité et structuration des objets

Le manifeste permet également d'**encapsuler des données liées à un ensemble d'objets**,

dont la **structuration interne** (ordre, hiérarchie, relations) est **définie par le manifeste lui-même**.

Ex :

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)
  - Listes ([])
  - Dictionnaire ({})

Exemple de structure 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 - Suite

Métadonnées institutionnelles
Le **format de manifeste IIIF** est **identique pour toutes les institutions**, mais **le contenu et la description des métadonnées** varient selon les **pratiques et les besoins** de chaque établissement.
Chaque institution définit ses propres champs descriptifs pour valoriser ses collections.Ex : https://media.getty.edu/iiif/manifest/1c76b1df-5f43-4340-bf2a-a9200d92142a

Les manifestes d'annotations
Il existe également des **manifestes d'annotations**, qui ne contiennent **aucune image** mais servent de **conteneurs d'informations complémentaires** :

- descriptions,
- transcriptions,
- commentaires, etc.

Intérêts de séparer annotations et objets :

- **Alléger les données** et **réduire le temps de chargement**,
- **Faciliter la mise à jour** ou la modification des annotations,
- **Éviter la diffusion d'informations inutiles** pour certains usages. Ex : https://iiif.bodleian.ox.ac.uk/iiif/annotationlist/3da059d4-e824-4082-832d-7ee2705fb9af.json

Structure des manifestes - Principes

JSON-LD : un format pour les données liées

JSON-LD (*JavaScript Object Notation for Linked Data*) est une **extension du format JSON** qui permet d'intégrer des **données liées** (*Linked Data*) dans des documents structurés.
Il est conçu pour :

- rendre les données **compréhensibles et exploitables par les machines**,
- **faciliter l'interopérabilité** des données sur le Web,
- permettre la **connexion entre différentes ressources** issues de systèmes distincts.

Les mots-clés réservés

Dans un document **JSON-LD**, certaines clés sont précédées du symbole **@**.
Ces clés sont des **mots-clés réservés** ayant une signification particulière dans le cadre des données liées.
Exemples :

- @context → décrit le cadre sémantique du document,
- @id → identifiant unique de la ressource,
- @type → type de l'objet décrit.

Ces conventions permettent de **distinguer les métadonnées normalisées** des **clés personnalisées** propres à chaque manifeste, garantissant ainsi une **interopérabilité maximale** entre institutions et outils.

Structure des manifestes - Composants principaux

@context : Spécifie le **contexte JSON-LD**, c'est-à-dire la définition des termes utilisés dans le document.

- Le contexte établit des **correspondances entre les termes locaux** du manifeste et les **concepts du Web sémantique**, garantissant ainsi la **compréhension et l'interopérabilité** des données entre systèmes.

@id : Représente **l'identifiant unique** d'une ressource sur le Web.

- Dans un manifeste IIIF, il s'agit le plus souvent de l'**URL** identifiant un objet particulier (manuscrit, image, vidéo, etc.).
- Cet identifiant permet de **référencer ou d'accéder directement à la ressource**.

label : Correspond au **titre** ou au **nom de la ressource**.

- Il s'agit généralement du texte affiché dans les visualiseurs IIIF ou dans les interfaces utilisateurs.

@type : Indique le **type de la ressource**.
Cela précise la **nature de l'objet** manipulé, par exemple :

- sc:Manifest → pour un manifeste,
- sc:Canvas → pour une toile (page, image, etc.).
  Ce champ aide les applications à **interpréter correctement** la structure et la fonction de chaque élément.

metadata : Contient la **liste des métadonnées supplémentaires** décrivant la ressource.

- Leur **structure et exhaustivité** varient selon les **institutions** et les **objets numériques** concernés.

sequence : Définit la **liste ordonnée des éléments** constituant l'œuvre (par exemple, les **pages** ou **folios** d'un manuscrit).

- Elle détermine **l'ordre de lecture ou d'affichage** dans les visualiseurs IIIF.

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

Définition

L'**API Image** est l'**interface de diffusion** des images numériques en **haute résolution** au sein du cadre IIIF.
Elle fournit un **protocole d'accès normalisé** permettant de **consulter, manipuler et afficher** des images à distance, de manière cohérente et interopérable.

Apports de l'API Image

- Une **syntaxe d'URL standardisée** pour accéder à une image et la manipuler à distance (recadrage, rotation, changement de taille, etc.).
- La fourniture d'**informations techniques détaillées** sur l'image, permettant à un visualiseur IIIF d'**adapter l'affichage** selon le contexte (taille d'écran, zoom, niveau de détail…).
- Des **URL modifiables manuellement**, facilitant la **personnalisation des images** selon les besoins de l'utilisateur.
- Des **URL optimisées pour la mise en cache**, garantissant des **performances rapides** et une **réutilisation efficace** des ressources.

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 `requests` et `pathlib`

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.

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

Qu'est-ce que pathlib ?
pathlib est un **module standard de Python** qui permet de **manipuler les chemins de fichiers et de répertoires** sous forme d'**objets** plutôt que de simples chaînes de caractères.

Avantages clés
- API orientée objet, plus lisible et expressive
- Compatible Windows / macOS / Linux sans adaptation
- Intégration facile avec d'autres bibliothèques (pandas, shutil, etc.)
- Code plus propre et plus sûr que os.path

Création et propriétés d'un chemin

```
from pathlib import Path
p = Path("data/fichier.txt")
```

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"

Lecture et écriture de fichiers
p.read_text() → lit le contenu texte d'un fichier
p.write_text("Nouveau contenu") → écrit du texte
Existe aussi en version binaire : read_bytes() / write_bytes()

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

`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")`

`Img.show()` : Affiche l'image dans une fenêtre.

- Idéal pour une prévisualisation rapide.
- **Ex :** `img.show()`

`Img.save()` : Enregistre l'image dans un fichier avec le format spécifié.

- **Ex :** `img.save("cheminvers/new_img.jpg", "JPEG")`

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.

Notions clés à retenir -IIIF

IIIF – Un cadre d'interopérabilité

- **IIIF (International Image Interoperability Framework)** : standard ouvert pour **diffuser, annoter et partager** des images et objets numériques sur le Web.
- Objectif : **décloisonner les collections** et offrir un **accès commun** aux ressources patrimoniales.

API Présentation et Manifeste IIIF

- L'**API Présentation** décrit la **structure et les métadonnées** d'un objet numérique (via un **Manifeste JSON-LD**).
- Le **Manifeste** sert d'**enveloppe virtuelle** regroupant images, séquences, annotations et métadonnées.
- Format **JSON-LD** : permet d'exprimer des **données liées (Linked Data)** et d'assurer l'**interopérabilité** entre institutions.

API Image

- Fournit un **accès normalisé** aux images en haute résolution.
- **URL paramétrable** : région, taille, rotation, qualité, format.
- Favorise la **manipulation à distance**, la **mise en cache** et l'intégration dans des visualiseurs IIIF.

Notions clés à retenir - Gestion de fichiers et téléchargements

Gérer les téléchargements d'images

- La bibliothèque **requests** permet de **récupérer des ressources HTTP** (comme des images IIIF) simplement et efficacement. Elle gère les **requêtes GET**, le **contenu binaire** et les **exceptions réseau**.

Gestion de fichiers et chemins

- **pathlib** : module standard pour **manipuler les chemins de fichiers** sous forme d'objets.
- Facilite la **création, vérification et lecture/écriture** de fichiers de manière **portable et lisible**.

Manipulation d'images

- La bibliothèque **PIL (Pillow)** permet de **traiter et transformer des images** :
  - ouverture, redimensionnement, conversion de format, filtres, recadrage.
- Complète l'usage d'IIIF pour des **traitements locaux** et des **exports personnalisés**.