***PROJET 4 : TOURNOI D’ECHEC***

**📌 Résumé et Points Clés : Programmation Orientée Objet (POO)**

**1️ les Classes et les Objets**

* **Une classe est un modèle (recette)** permettant de créer des objets.
* **Un objet est une instance d’une classe** avec des caractéristiques propres.
* **Exemple :**
  + Une classe Cake (gâteau) définit une structure avec des attributs (flavor) et des méthodes (cut\_in\_parts()).
  + Chaque gâteau créé avec cette recette est un objet spécifique (Cake("chocolat") et Cake("banane")).

**2️ État et Comportement**

* **État = les variables (attributs) stockées dans l’objet.**
* **Comportement = les méthodes qui définissent ce que l’objet peut faire.**
  + Exemple :

class Cake:

def \_\_init\_\_(self, flavor):

self.flavor = flavor # Attribut (état)

def cut\_in\_parts(self):

print(f"Le gâteau {self.flavor} est coupé en parts.") # Méthode (comportement)

chocolat\_cake = Cake("chocolat")

chocolat\_cake.cut\_in\_parts() # Utilisation d’une méthode

**3️ Différence entre une Fonction et une Méthode**

* **Fonction** : indépendante (ex. print(), sum()).
* **Méthode** : associée à une classe et utilisée via un objet (ex. obj.methode()).

**4️ Pourquoi Utiliser la POO ?**

* **Organisation et structure** : au lieu de gérer des variables et fonctions séparées, tout est regroupé dans des classes.
* **Réutilisabilité** : on peut créer plusieurs objets avec le même modèle.
* **Facilité de maintenance** : le code est plus lisible et mieux organisé.

**✏️ Exercice : Identifiez les Classes**

Pour chaque situation, identifiez une **classe**, ses **attributs** et ses **méthodes**.

**🛠 Boîte à outils**

* **Classe :** Toolbox
* **Attributs :** tools (liste des outils)
* **Méthodes :** add\_tool(), remove\_tool()

**🔨 Marteau**

* **Classe :** Hammer
* **Attributs :** color
* **Méthodes :** use\_hammer(), remove\_nail(), paint(new\_color)

**🪛 Tournevis**

* **Classe :** Screwdriver
* **Attributs :** size
* **Méthodes :** tighten\_screw(), loosen\_screw()

**📌 Résumé et Points Clés : Écriture d’une Classe Python**

**1️⃣ Déclaration d’une Classe**

* Utilisation du mot-clé **class** suivi du **nom de la classe** (convention : **CapitalizedCase**).
* Les attributs et méthodes sont définis à l’intérieur avec une **indentation** (4 espaces).
* **Exemple :**

python

Copier

class Rectangle:

width = 3 # Attribut de classe

height = 2

def calculate\_area(self): # Méthode

return self.width \* self.height

* **self** permet d’accéder aux attributs de l’instance.

**2️⃣ Différents Types d’Attributs**

* **Attributs de classe** : Partagés entre toutes les instances.
* **Attributs d’instance** : Propre à chaque objet créé à partir de la classe.
* **Attributs statiques** : Indépendants de la classe et de ses instances.

**3️⃣ Le Constructeur \_\_init\_\_**

* Méthode spéciale appelée lors de la création d’un objet.
* Permet d’initialiser des **attributs d’instance**.
* **Exemple :**

python

Copier

class Rectangle:

def \_\_init\_\_(self, length, width, color="red"):

self.length = length

self.width = width

self.color = color # Valeur par défaut = "red"

* **Les valeurs définies dans \_\_init\_\_ ne sont pas partagées** entre les instances.
* On peut inclure des **paramètres optionnels** (ex. color="red").

**✏️ Exercice : Implémentez les Classes**

**📌 Classe Toolbox (Boîte à Outils)**

* **Attributs :** tools (liste des outils).
* **Méthodes :** add\_tool(), remove\_tool().

**📌 Classe Hammer (Marteau)**

* **Attributs :** color.
* **Méthodes :** use\_hammer(), remove\_nail(), paint(new\_color).

**📌 Classe Screwdriver (Tournevis)**

* **Attributs :** size.
* **Méthodes :** tighten\_screw(), loosen\_screw().

Voici l’implémentation en Python avec des **constructeurs** :

Version class par class :

class Toolbox:

"""Représente une boîte à outils contenant des outils."""

def \_\_init\_\_(self):

self.tools = [] # Liste des outils

def add\_tool(self, tool):

"""Ajoute un outil à la boîte."""

self.tools.append(tool)

def remove\_tool(self, tool):

"""Retire un outil de la boîte."""

if tool in self.tools:

self.tools.remove(tool)

class Hammer:

"""Représente un marteau avec une couleur."""

def \_\_init\_\_(self, color):

self.color = color

def use\_hammer(self):

"""Utiliser le marteau pour enfoncer un clou."""

print("BAM ! Clou enfoncé.")

def remove\_nail(self):

"""Utiliser le marteau pour retirer un clou."""

print("Clou retiré.")

def paint(self, new\_color):

"""Changer la couleur du marteau."""

self.color = new\_color

class Screwdriver:

"""Représente un tournevis avec une taille."""

def \_\_init\_\_(self, size):

self.size = size

def tighten\_screw(self):

"""Serrer une vis."""

print("Vis serrée.")

def loosen\_screw(self):

"""Desserrer une vis."""

print("Vis desserrée.")

version openclassrooms avec « pass » :

"""Définit des classes d'outils."""

class ToolBox:

"""Boite à outils."""

def \_\_init\_\_(self):

"""Initialise les outils."""

self.tools = []

def add\_tool(self, tool):

"""Ajoute un outil."""

pass

def remove\_tool(self, tool):

"""Enleve un outil."""

pass

class Screwdriver:

"""Tournevis."""

def \_\_init\_\_(self, size):

"""Initialise la taille."""

self.size = size

def tighten(self, screw):

"""Serrer une vis."""

pass

def loosen(self, screw):

"""Desserre une vis."""

pass

class Hammer:

"""Marteau."""

def \_\_init\_\_(self, color="red"):

"""Initialise la couleur."""

self.color = color

def paint(self, color):

"""Paint le marteau."""

pass

def hammer\_in(self, nail):

"""Enfonce un clou."""

pass

def remove(self, nail):

"""Enleve un clou."""

Pass

**📌 Résumé et Points Clés : Création et Utilisation d’Objets en Python**

**1️⃣ Instanciation d’un Objet**

* Un **objet** est une **instance** d’une classe.
* Pour **créer un objet**, on appelle le **constructeur** de la classe (\_\_init\_\_).
* **Syntaxe :**

python

CopierModifier

rectangle = Rectangle(5, 3) # Création d'un objet Rectangle avec longueur 5 et largeur 3

* Il est recommandé d’utiliser des **arguments nommés** pour plus de clarté :

python

CopierModifier

rect = Rectangle(length=5, width=3, color="blue")

**2️⃣ Modification d’un Objet**

* **Accéder et modifier un attribut** :

python

CopierModifier

print(rectangle.length) # Affiche la longueur du rectangle

rectangle.color = "yellow" # Change la couleur du rectangle

* **Appeler une méthode d’un objet** :

python

CopierModifier

area = rectangle.calculate\_area() # Calcule l’aire du rectangle

print(area) # Affiche l’aire

**3️⃣ Les Différents Types d’Attributs**

| **Type d’Attribut** | **Définition** | **Accessible sans instanciation ?** |
| --- | --- | --- |
| **Attribut d’instance** | Stocké dans self, propre à chaque objet | ❌ Non |
| **Attribut de classe** | Défini dans la classe, partagé entre toutes les instances | ✅ Oui |
| **Attribut statique** | Indépendant de la classe et des objets, souvent utilisé pour des constantes | ✅ Oui |

* **Exemple :**

python

CopierModifier

class Bird:

species = "Oiseau" # Attribut de classe partagé

def \_\_init\_\_(self, name):

self.name = name # Attribut d’instance propre à chaque oiseau

@classmethod

def get\_species(cls):

return cls.species # Méthode de classe

@staticmethod

def get\_definition():

return "Un oiseau est un animal vertébré à plumes."

print(Bird.get\_species()) # Accès sans instanciation

print(Bird.get\_definition()) # Accès direct à la méthode statique

**✏️ Exercice : Instanciation et Manipulation d’Objets**

**📌 Étapes à Suivre :**

1. **Créer une boîte à outils (ToolBox)**, un tournevis (Screwdriver), et un marteau (Hammer).
2. **Ajouter** le tournevis et le marteau dans la boîte à outils.
3. **Créer une vis (Screw)**, la **serrer avec le tournevis**, puis afficher son état avant/après.
4. **Créer un clou (Nail)**, l'**enfoncer avec le marteau**, puis afficher son état avant/après.

**🛠 Implémentation en Python**

python

CopierModifier

# Instanciation des objets

toolbox = Toolbox()

screwdriver = Screwdriver(size=5)

hammer = Hammer(color="blue")

# Ajout des outils à la boîte à outils

toolbox.add\_tool(screwdriver)

toolbox.add\_tool(hammer)

# Création et manipulation d'une vis

screw = Screw()

print(screw) # Avant serrage

screwdriver.tighten(screw)

print(screw) # Après serrage

# Création et manipulation d’un clou

nail = Nail()

print(nail) # Avant enfoncement

hammer.hammer\_in(nail)

print(nail) # Après enfoncement

version openclassrooms :

"""Ici, nous allons serrer des vis et taper sur des clous!"""

class ToolBox:

"""Boite à outils."""

def \_\_init\_\_(self):

"""Initialise les outils."""

self.tools = []

def add\_tool(self, tool):

"""Ajoute un outil."""

self.tools.append(tool)

def remove\_tool(self, tool):

"""Enleve un outil."""

index = self.tools.index(tool)

del self.tools[index]

class Screwdriver:

"""Tournevis."""

def \_\_init\_\_(self, size=3):

"""Initialise la taille."""

self.size = size

def tighten(self, screw):

"""Serrer une vis."""

screw.tighten()

def loosen(self, screw):

"""Desserre une vis."""

screw.loosen()

def \_\_repr\_\_(self):

"""Représentation de l'objet."""

return f"Tournevis de taille {self.size}"

class Hammer:

"""Marteau."""

def \_\_init\_\_(self, color="red"):

"""Initialise la couleur."""

self.color = color

def paint(self, color):

"""Paint le marteau."""

self.color = color

def hammer\_in(self, nail):

"""Enfonce un clou."""

nail.nail\_in()

def remove(self, nail):

"""Enleve un clou."""

nail.remove()

def \_\_repr\_\_(self):

"""Représentation de l'objet."""

return f"Marteau de couleur {self.color}"

class Screw:

"""Vis."""

MAX\_TIGHTNESS = 5

def \_\_init\_\_(self):

"""Initialise son degré de serrage."""

self.tightness = 0

def loosen(self):

"""Déserre le vis."""

if self.tightness > 0:

self.tightness -= 1

def tighten(self):

"""Serre le vis."""

if self.tightness < self.MAX\_TIGHTNESS:

self.tightness += 1

def \_\_str\_\_(self):

"""Retourne une forme lisible de l'objet."""

return "Vis avec un serrage de {}".format(self.tightness)

class Nail:

"""Clou."""

def \_\_init\_\_(self):

"""Initialise son statut "dans le mur"."""

self.in\_wall = False

def nail\_in(self):

"""Enfonce le clou dans un mur."""

if not self.in\_wall:

self.in\_wall = True

def remove(self):

"""Enlève le clou du mur."""

if self.in\_wall:

self.in\_wall = False

def \_\_str\_\_(self):

"""Retourne une forme lisible de l'objet."""

wall\_state = "dans le mur" if self.in\_wall else "hors du mur"

return f"Clou {wall\_state}."

# Instanciez une boîte à outils, un tournevis, et un marteau.

hammer = Hammer()

screwdriver = Screwdriver()

toolbox = ToolBox()

# Placez le marteau et le tournevis dans la boîte à outils.

toolbox.add\_tool(hammer)

toolbox.add\_tool(screwdriver)

# Instanciez une vis, et serrez-la avec le tournevis.

# Affichez la vis avant après avoir été serrée.

screw = Screw()

print(screw)

screwdriver.tighten(screw)

print(screw)

# Instanciez un clou, puis enfoncez-le avec le marteau.

# Affichez le clou avant et après avoir été enfoncé.

nail = Nail()

print(nail)

hammer.hammer\_in(nail)

print(nail)

# --------------------------------------------------------------

# Que pouvez-vous faire d’autre avec ces classes et ces objets ?

# enlever un outil

print("outils dans la boîte:", toolbox.tools)

toolbox.remove\_tool(hammer)

print("on a enlevé le marteau")

print("outils dans la boîte:", toolbox.tools)

# désserrer la vis

screwdriver.loosen(screw)

print(screw)

# enlever le clou

hammer.remove(nail)

print(nail)

# repeindre le marteau

hammer.paint("yellow")

print(hammer)

Voici les réponses au quiz avec explication pour chaque question :

**Question 1**

**Qu’est-ce qu’une classe ?**  
✅ **Un modèle pour un objet.**

**Explication :** Une classe est un modèle permettant de créer des objets en programmation orientée objet (POO).

**Question 2**

**Voici la différence entre une méthode et une fonction :**  
✅ **Une méthode fait partie d’une classe, ce qui n’est pas le cas d’une fonction.**

**Explication :** Une méthode est liée à une classe et s’applique à une instance de cette classe. Une fonction, en revanche, est indépendante.

**Question 3**

**Parmi ces signatures de constructeur, laquelle est correcte ?**  
✅ **def \_\_init\_\_(self, name, date\_of\_birth)**

**Explication :**

* \_\_init\_\_ est le nom du constructeur.
* self est le premier paramètre obligatoire pour faire référence à l’instance de l’objet.
* Les autres paramètres name et date\_of\_birth sont bien séparés par des virgules.

**Question 4**

**Une classe Book a un constructeur avec cette signature :**  
def \_\_init\_\_(self, price, title="")  
**Quelles déclarations sont valides ?**  
✅ **book = Book(5)**  
✅ **book = Book(1, "Les Mille et Une Nuits")**  
✅ **book = Book(price=4, title="Antigone")**  
❌ book = Book(self) (incorrect, car self ne doit pas être passé explicitement)  
❌ book = Book(self, 3) (même raison que ci-dessus)

**Question 5**

**Parmi ces définitions de méthodes area(), lesquelles sont correctes ?**  
✅ **def area(self): return self.length \* self.width**  
✅ **def area(this): return this.length \* this.width** (self est une convention, this fonctionnerait aussi)  
❌ def area(): return self.length \* self.width (manque self en paramètre)  
❌ def area(self): return length \* width (manque self. avant length et width)

**Question 6**

✅ **Faux.**

**Explication :** Contrairement à d'autres langages comme Java, Python ne nécessite pas de déclaration explicite du type des variables dans une classe.

**Question 7**

✅ **L’attribut de classe est global à la classe. Il n’est pas nécessaire d’instancier la classe pour y accéder.**

**Explication :**

* Un **attribut de classe** est partagé entre toutes les instances et accessible directement via la classe.
* Un **attribut d’instance** est propre à chaque objet et nécessite une instanciation.

**Question 8**

**Quel sera le résultat de ce code ?**

python

CopierModifier

alice = User("Alice", "alice@example.com")

alice.email = "contact@alice.com"

✅ **L’adresse e-mail d’Alice va changer.**

**Explication :**

* Les objets Python ne sont **pas en lecture seule**.
* On peut modifier un attribut d’instance après la création d’un objet.

**Question 9**

✅ **Faux.**

**Explication :**

* Si un constructeur \_\_init\_\_ n’est pas défini, Python utilisera un **constructeur par défaut**.
* On peut donc instancier une classe même si elle ne définit pas explicitement \_\_init\_\_.

**Question 10**

**Quelle proposition permet de connecter un PC et un écran à la prise intelligente ?**  
✅ **Ce code est correct :**

python

CopierModifier

plug = SmartPlug()

pc = Device("pc")

screen = Device("écran")

plug.connect(pc)

plug.connect(screen)

plug.display\_connected\_devices()

❌ SmartPlug.connect(pc, screen) est incorrect, car connect est une méthode d'instance et doit être appelée sur un objet (plug).  
❌ pc = PC() et screen = Screen() sont incorrects car PC et Screen ne sont pas définis dans le code fourni.

**🎯 Résumé :**

| **Question** | **Réponse Correcte** |
| --- | --- |
| **1** | Un modèle pour un objet |
| **2** | Une méthode fait partie d’une classe |
| **3** | def \_\_init\_\_(self, name, date\_of\_birth) |
| **4** | Book(5), Book(1, "Les Mille et Une Nuits"), Book(price=4, title="Antigone") |
| **5** | def area(self): return self.length \* self.width et def area(this): return this.length \* this.width |
| **6** | Faux |
| **7** | L’attribut de classe est global à la classe |
| **8** | L’adresse e-mail d’Alice va changer |
| **9** | Faux |
| **10** | plug = SmartPlug(); plug.connect(pc); plug.connect(screen); plug.display\_connected\_devices() |

**📌 Résumé et Points Clés : L’Héritage en Python**

**1️⃣ Qu’est-ce que l’héritage ?**

L’héritage permet de **créer une classe enfant (sous-classe)** qui **hérite** des attributs et méthodes d’une **classe parent (superclasse)**.

* **Exemple :**

python

CopierModifier

class Film:

def \_\_init\_\_(self, name):

self.name = name

def watch(self):

print(f"Regarder {self.name}")

class FilmCassette(Film): # FilmCassette hérite de Film

def \_\_init\_\_(self, name, magnetic\_tape=True):

super().\_\_init\_\_(name) # Appel du constructeur de Film

self.magnetic\_tape = magnetic\_tape

def rewind(self):

print(f"Rembobiner {self.name}")

* **Une sous-classe peut :**
  + **Réutiliser** les attributs et méthodes de la classe parent.
  + **Ajouter** ses propres attributs et méthodes.
  + **Modifier (surcharger)** les méthodes de la classe parent.

**2️⃣ Pourquoi utiliser l’héritage ?**

**✅ Réutilisabilité**

* Évite la duplication de code.
* Facilite les mises à jour et les modifications.

**✅ Extensibilité**

* Ajout de nouvelles fonctionnalités **sans modifier** la classe parent.

**✅ Organisation et Modélisation**

* Structure plus claire des classes.
* Facilite la compréhension et la maintenance du code.

**3️⃣ ✏️ Exercice : Identifier les Classes et Héritages**

**📌 Situation : Forum sur les Gâteaux**

Les utilisateurs peuvent :

* **S’inscrire et se connecter** → Classe User
* **Créer et répondre à des discussions** → Classe Post
* **Créer des fils de discussion** → Classe Thread
* **Attacher des fichiers aux posts** → Classe Attachment
  + Certains sont des **images** → Classe ImageAttachment
* **Modérateurs** qui peuvent éditer et supprimer des posts → Classe Moderator (hérite de User)

**📌 Diagramme des classes**

scss

CopierModifier

User

│──> Moderator (hérite de User)

│

Thread

│──> Post (contient un User et un contenu)

│ │──> PostWithAttachment (hérite de Post)

│ │──> ImageAttachment (hérite d’Attachment)

│

Attachment

│──> ImageAttachment

**📌 Implémentation en Python**

python

CopierModifier

class User:

"""Représente un utilisateur du forum."""

def \_\_init\_\_(self, username, email):

self.username = username

self.email = email

def login(self):

print(f"{self.username} est connecté.")

class Moderator(User):

"""Modérateur qui peut modifier ou supprimer des posts."""

def edit\_post(self, post, new\_content):

post.content = new\_content

print(f"Le post a été modifié par {self.username}.")

def delete\_post(self, post):

print(f"Le post '{post.content}' a été supprimé par {self.username}.")

class Thread:

"""Représente un fil de discussion."""

def \_\_init\_\_(self, title, creator):

self.title = title

self.creator = creator

self.posts = []

def add\_post(self, post):

self.posts.append(post)

class Post:

"""Représente un post dans un fil de discussion."""

def \_\_init\_\_(self, user, content):

self.user = user

self.content = content

def display(self):

print(f"{self.user.username} a écrit : {self.content}")

class Attachment:

"""Représente un fichier attaché à un post."""

def \_\_init\_\_(self, filename):

self.filename = filename

class ImageAttachment(Attachment):

"""Spécialisation pour les images."""

def \_\_init\_\_(self, filename, resolution):

super().\_\_init\_\_(filename)

self.resolution = resolution

class PostWithAttachment(Post):

"""Post avec un fichier attaché."""

def \_\_init\_\_(self, user, content, attachment):

super().\_\_init\_\_(user, content)

self.attachment = attachment

def display(self):

super().display()

print(f"📎 Fichier attaché : {self.attachment.filename}")

réponse très proche de celle d’OpenClassrooms !

comparaison et quelques ajustements pour bien coller à leur modèle.

**📌 Correction et Validation des Classes**

**✅ Classe File et Image**

* **Classe File** : représente un fichier attaché.
* **Classe Image** : hérite de File, spécifique aux images.

| **Classe** | **État (Attributs)** | **Comportement (Méthodes)** |
| --- | --- | --- |
| File | name, size | display() |
| Image | (hérite de File) | (aucune méthode ajoutée) |

python

CopierModifier

class File:

def \_\_init\_\_(self, name, size):

self.name = name

self.size = size

def display(self):

print(f"Affichage du fichier : {self.name} ({self.size} Mo)")

class Image(File):

pass # Hérite de File sans modification

**✅ Classe User et Moderator**

* **Classe User** : gère l’inscription et l’interaction.
* **Classe Moderator** : peut éditer et supprimer des messages.

| **Classe** | **État (Attributs)** | **Comportement (Méthodes)** |
| --- | --- | --- |
| User | username, password | login(), post(), make\_thread() |
| Moderator | (hérite de User) | edit(), delete() |

python

CopierModifier

class User:

def \_\_init\_\_(self, username, password):

self.username = username

self.password = password

def login(self):

print(f"{self.username} s'est connecté.")

def post(self, thread, content):

thread.add\_post(Post(self, content))

def make\_thread(self, title, content):

return Thread(title, self)

class Moderator(User):

def edit(self, post, content):

post.content = content

print(f"Le post a été modifié par {self.username}.")

def delete(self, thread, post):

thread.remove\_post(post)

print(f"Le post a été supprimé par {self.username}.")

**✅ Classe Post et FilePost**

* **Classe Post** : contient du texte et un utilisateur.
* **Classe FilePost** : ajoute un fichier joint.

| **Classe** | **État (Attributs)** | **Comportement (Méthodes)** |
| --- | --- | --- |
| Post | user, time\_posted, content | display() |
| FilePost | (hérite de Post), file | (hérite de display()) |

python

CopierModifier

from datetime import datetime

class Post:

def \_\_init\_\_(self, user, content):

self.user = user

self.content = content

self.time\_posted = datetime.now()

def display(self):

print(f"{self.user.username} a écrit : {self.content} ({self.time\_posted})")

class FilePost(Post):

def \_\_init\_\_(self, user, content, file):

super().\_\_init\_\_(user, content)

self.file = file

def display(self):

super().display()

print(f"📎 Fichier attaché : {self.file.name}")

**✅ Classe Thread**

* Contient un **titre** et une liste de posts.

| **Classe** | **État (Attributs)** | **Comportement (Méthodes)** |
| --- | --- | --- |
| Thread | title, posts | add\_post(), remove\_post() |

python

CopierModifier

class Thread:

def \_\_init\_\_(self, title):

self.title = title

self.posts = []

def add\_post(self, post):

self.posts.append(post)

def remove\_post(self, post):

if post in self.posts:

self.posts.remove(post)

**✅ Comparaison avec OpenClassrooms**

| **Élément** | **OpenClassrooms** | **Ma Version** |
| --- | --- | --- |
| **Fichiers** | File, Image | ✅ Identique |
| **Utilisateurs** | User, Moderator | ✅ Identique |
| **Posts** | Post, FilePost | ✅ Identique |
| **Threads** | Thread | ✅ Identique |

Tu es donc **totalement aligné** avec la solution officielle ! 🎉

**🚀 Prochaine Étape**

1. **Créer et tester des objets** avec ces classes.
2. **Passer à la surcharge des méthodes en Python** pour modifier le comportement hérité.

**Résumé et Points Clés : Écriture d’une Sous-classe en Python**

**1️⃣ Définition d’une Sous-classe**

Une sous-classe est définie en mettant le **nom de la classe parent entre parenthèses** :

python

CopierModifier

class FilmCassette(Film): # Hérite de Film

"""Un film en cassette !"""

def \_\_init\_\_(self, name):

"""Initialise le nom et la bande magnétique."""

self.name = name

self.magnetic\_tape = True

def rewind(self):

"""Rembobine le film."""

print("C'est long à rembobiner !")

self.magnetic\_tape = True

**2️⃣ L’Héritage par Défaut**

Si une classe n’a pas de parent explicite, elle hérite automatiquement de **object**, qui est la base de toutes les classes en Python.

**3️⃣ Héritage Multiple**

Python permet l’héritage multiple en séparant les classes parents par des virgules :

python

CopierModifier

class SousClasse(ParentUn, ParentDeux):

pass

⚠ **À utiliser avec précaution** pour éviter les conflits et la complexité inutile.

**✏️ Exercice : Implémentation des Classes du Forum**

Nous allons réécrire les classes en utilisant **l’héritage**.

**✅ Classe File et Image**

python

CopierModifier

class File:

"""Représente un fichier attaché à un post."""

def \_\_init\_\_(self, name, size):

self.name = name

self.size = size

def display(self):

"""Affiche le fichier."""

print(f"Fichier : {self.name} ({self.size} Mo)")

class Image(File):

"""Spécialisation des fichiers pour les images."""

pass # Aucun changement nécessaire

**✅ Classe User et Moderator**

python

CopierModifier

class User:

"""Représente un utilisateur du forum."""

def \_\_init\_\_(self, username, password):

self.username = username

self.password = password

def login(self):

print(f"{self.username} s'est connecté.")

def post(self, thread, content):

thread.add\_post(Post(self, content))

def make\_thread(self, title, content):

return Thread(title, self)

class Moderator(User):

"""Un modérateur qui peut modifier ou supprimer des posts."""

def edit(self, post, content):

"""Modifie le contenu d'un post."""

post.content = content

print(f"Le post a été modifié par {self.username}.")

def delete(self, thread, post):

"""Supprime un post d'un thread."""

thread.remove\_post(post)

print(f"Le post a été supprimé par {self.username}.")

**✅ Classe Post et FilePost**

python

CopierModifier

from datetime import datetime

class Post:

"""Représente un post dans un fil de discussion."""

def \_\_init\_\_(self, user, content):

self.user = user

self.content = content

self.time\_posted = datetime.now()

def display(self):

"""Affiche le contenu du post."""

print(f"{self.user.username} ({self.time\_posted}) : {self.content}")

class FilePost(Post):

"""Un post qui contient un fichier attaché."""

def \_\_init\_\_(self, user, content, file):

super().\_\_init\_\_(user, content)

self.file = file

def display(self):

"""Affiche le post avec le fichier attaché."""

super().display()

print(f"📎 Fichier attaché : {self.file.name}")

**✅ Classe Thread**

python

CopierModifier

class Thread:

"""Représente un fil de discussion."""

def \_\_init\_\_(self, title, creator):

self.title = title

self.creator = creator

self.posts = []

def add\_post(self, post):

"""Ajoute un post au fil de discussion."""

self.posts.append(post)

def remove\_post(self, post):

"""Supprime un post du fil de discussion."""

if post in self.posts:

self.posts.remove(post)

Solution de openclassrooms :  
  
"""Définit les classes propres à notre forum. ;)"""

class File:

"""Fichier."""

def \_\_init\_\_(self, name, size):

"""Initialise le nom et la taille."""

self.name = name

self.size = size

def display(self):

"""Affiche le fichier."""

print(f"Fichier '{self.name}'.")

class ImageFile(File):

"""Fichier image.

Pas plus à ajouter pour l'instant !

"""

pass

class User:

"""Utilisateur."""

def \_\_init\_\_(self, username, password):

"""Initialise le nom d'utilisateur et le mot de passe."""

self.username = username

self.password = password

def login(self):

"""Connecte l'utilisateur."""

print(f"L'utilisateur {self.username} est connecté.")

def post(self, thread, content, file=None):

"""Poste un message dans un fil de discussion."""

if file:

post = FilePost(self, "aujourd'hui", content, file)

else:

post = Post(user=self, time\_posted="aujourd'hui", content=content)

thread.add\_post(post)

return post

def make\_thread(self, title, content):

"""Créé un nouveau fil de discussion."""

post = Post(self, "aujourd'hui", content)

return Thread(title, "aujourd'hui", post)

def \_\_str\_\_(self):

"""représentation de l'utilisateur."""

return self.username

class Moderator(User):

"""Utilisateur modérateur."""

def edit(self, post, content):

"""Modifie un message."""

post.content = content

def delete(self, thread, post):

"""Supprime un message."""

index = thread.posts.index(post)

del thread.posts[index]

class Post:

"""Message."""

def \_\_init\_\_(self, user, time\_posted, content):

"""Initialise l'utilisateur, la date et le contenu."""

self.user = user

self.time\_posted = time\_posted

self.content = content

def display(self):

"""Affiche le message."""

print(f"Message posté par {self.user} le {self.time\_posted}:")

print(self.content)

class FilePost(Post):

"""Message comportant un fichier."""

def \_\_init\_\_(self, user, time\_posted, content, file):

"""Initialise le fichier."""

self.user = user

self.time\_posted = time\_posted

self.content = content

self.file = file

def display(self):

"""Affiche le contenu et le fichier."""

super().display()

print("pièce jointe:")

self.file.display()

class Thread:

"""Fil de discussions."""

def \_\_init\_\_(self, title, time\_posted, post):

"""Initialise le titre, la date et les posts.

Attention ici: on commence par un seul post, celui du sujet.

Les réponses à ce post ne pourrons s'ajouter qu'ultérieurement.

En effet, on ne créé pas directement un nouveau fil avec des réponses. ;)

"""

self.title = title

self.time\_posted = time\_posted

self.posts = [post]

def display(self):

"""Affiche le fil de discussion."""

print("----- THREAD -----")

print(f"titre: {self.title}, date: {self.time\_posted}")

print()

for post in self.posts:

post.display()

print()

print("------------------")

def add\_post(self, post):

"""Ajoute un post."""

self.posts.append(post)

**Résumé et Points Clés : Surcharge des Méthodes en Python**

**1️⃣ Qu’est-ce que la Surcharge de Méthodes ?**

Lorsqu’une **classe enfant** redéfinit une méthode existante de la **classe parent**, la nouvelle version l’emporte.

* **Exemple :**

python

CopierModifier

class Shape:

def area(self):

return 0 # Méthode générique

class Square(Shape):

def \_\_init\_\_(self, length):

self.length = length

def area(self): # Surcharge de la méthode area()

return self.length \* self.length

✅ Square.area() redéfinit Shape.area().

**2️⃣ Classes Abstraites**

Une **classe abstraite** ne peut **pas être instanciée**, elle sert uniquement de modèle pour ses sous-classes.

* **On utilise ABC et @abstractmethod** :

python

CopierModifier

from abc import ABC, abstractmethod

class Shape(ABC):

@abstractmethod

def area(self):

pass # Oblige les sous-classes à implémenter cette méthode

✅ **Toutes les sous-classes de Shape doivent redéfinir area().**

**3️⃣ Utilisation de super()**

* **super()** permet d’appeler une méthode de la classe parent sans la copier.
* **Exemple avec une boisson et du café** :

python

CopierModifier

class Drink:

def \_\_init\_\_(self, price):

self.price = price

class Coffee(Drink):

prices = {"simple": 1, "serré": 1, "allongé": 1.5}

def \_\_init\_\_(self, type):

self.type = type

super().\_\_init\_\_(price=self.prices.get(type, 1)) # Appelle le constructeur de Drink

✅ Coffee hérite de Drink et passe le prix au constructeur parent.

**✏️ Exercice : Ajouter la Surcharge**

Nous allons :

1. **Créer une classe abstraite Post** avec @abstractmethod.
2. **Surcharger display() dans FilePost** pour afficher le fichier avant le texte.
3. **Surcharger display() dans Image** pour afficher "📷 [nom de l’image]".

**📌 Implémentation en Python**

python

CopierModifier

from abc import ABC, abstractmethod

class Post(ABC):

"""Classe abstraite pour représenter un post."""

def \_\_init\_\_(self, user, content):

self.user = user

self.content = content

@abstractmethod

def display(self):

"""Méthode d'affichage à implémenter dans les sous-classes."""

pass

class FilePost(Post):

"""Post contenant un fichier attaché."""

def \_\_init\_\_(self, user, content, file):

super().\_\_init\_\_(user, content)

self.file = file

def display(self):

"""Affiche d'abord le fichier, puis le contenu du post."""

print(f"📎 Fichier attaché : {self.file.name}")

print(f"{self.user.username} a écrit : {self.content}")

class File:

"""Représente un fichier attaché à un post."""

def \_\_init\_\_(self, name, size):

self.name = name

self.size = size

def display(self):

"""Affiche le fichier."""

print(f"Fichier : {self.name} ({self.size} Mo)")

class Image(File):

"""Fichier image, spécialisation de File."""

def display(self):

"""Affiche un message spécial pour les images."""

print(f"📷 Image : {self.name}")

**🚀 Tests**

python

CopierModifier

# Création d'objets

user = User("Alice", "password123")

image = Image("gateau.jpg", 2.5)

file\_post = FilePost(user, "Regardez ce gâteau !", image)

# Affichage

file\_post.display()

image.display()

✅ **Résultat attendu :**

arduino

CopierModifier

📎 Fichier attaché : gateau.jpg

Alice a écrit : Regardez ce gâteau !

📷 Image : gateau.jpg

Solution openclassrooms :

"""Définit les classes propres à notre forum. ;)"""

from abc import ABC

class File(ABC):

"""Fichier."""

def \_\_init\_\_(self, name, size):

"""Initialise le nom et la taille."""

self.name = name

self.size = size

def display(self):

"""Affiche le fichier."""

pass

class ImageFile(File):

"""Fichier image."""

def display(self):

"""Affiche l'image."""

print(f"Fichier image '{self.name}'.")

class User:

"""Utilisateur."""

def \_\_init\_\_(self, username, password):

"""Initialise le nom d'utilisateur et le mot de passe."""

self.username = username

self.password = password

def login(self):

"""Connecte l'utilisateur."""

print(f"L'utilisateur {self.username} est connecté.")

def post(self, thread, content, file=None):

"""Poste un message dans un fil de discussion."""

if file:

post = FilePost(self, "aujourd'hui", content, file)

else:

post = Post(user=self, time\_posted="aujourd'hui", content=content)

thread.add\_post(post)

return post

def make\_thread(self, title, content):

"""Créé un nouveau fil de discussion."""

post = Post(self, "aujourd'hui", content)

return Thread(title, "aujourd'hui", post)

def \_\_str\_\_(self):

"""représentation de l'utilisateur."""

return self.username

class Moderator(User):

"""Utilisateur modérateur."""

def edit(self, post, content):

"""Modifie un message."""

post.content = content

def delete(self, thread, post):

"""Supprime un message."""

index = thread.posts.index(post)

del thread.posts[index]

class Post:

"""Message."""

def \_\_init\_\_(self, user, time\_posted, content):

"""Initialise l'utilisateur, la date et le contenu."""

self.user = user

self.time\_posted = time\_posted

self.content = content

def display(self):

"""Affiche le message."""

print(f"Message posté par {self.user} le {self.time\_posted}:")

print(self.content)

class FilePost(Post):

"""Message comportant un fichier."""

def \_\_init\_\_(self, user, time\_posted, content, file):

"""Initialise le fichier."""

super().\_\_init\_\_(user, time\_posted, content)

self.file = file

def display(self):

"""Affiche le contenu et le fichier."""

super().display()

print("pièce jointe:")

self.file.display()

class Thread:

"""Fil de discussions."""

def \_\_init\_\_(self, title, time\_posted, post):

"""Initialise le titre, la date et les posts.

Attention ici: on commence par un seul post, celui du sujet.

Les réponses à ce post ne pourrons s'ajouter qu'ultérieurement.

En effet, on ne créé pas directement un nouveau fil avec des réponses. ;)

"""

self.title = title

self.time\_posted = time\_posted

self.posts = [post]

def display(self):

"""Affiche le fil de discussion."""

print("----- THREAD -----")

print(f"titre: {self.title}, date: {self.time\_posted}")

print()

for post in self.posts:

post.display()

print()

print("------------------")

def add\_post(self, post):

"""Ajoute un post."""

self.posts.append(post)

**Résumé et Points Clés : Hiérarchie d’Héritage et Héritage Multiple en Python**

**1️⃣ Hiérarchie d’Héritage**

* Une **classe parent** peut elle-même **avoir un parent**.
* Une **classe enfant hérite de toutes les classes parents** dans la hiérarchie.
* Exemple :

python

CopierModifier

class Film:

def \_\_init\_\_(self, name):

self.name = name

def watch(self):

print(f"Regarder {self.name}")

class FilmCassette(Film): # Hérite de Film

def rewind(self):

print("Rembobinage du film")

class FilmCassetteVHS(FilmCassette): # Hérite de FilmCassette

def adjust\_tracking(self):

print("Ajustement du suivi VHS")

✅ FilmCassetteVHS **hérite de FilmCassette, qui hérite de Film**.

**2️⃣ Héritage Multiple**

* Une classe peut **hériter de plusieurs parents simultanément**.
* **Problème potentiel** : Si deux parents définissent la **même méthode**, quelle version sera utilisée ?
* **Python utilise le MRO (Method Resolution Order)** pour décider l’ordre des recherches de méthodes.
* Exemple :

python

CopierModifier

class Cat:

def meow(self):

print("Meow!")

class Talker:

def say(self, to\_say):

print(to\_say)

class TalkingCat(Cat, Talker): # Hérite des deux classes

pass

salem = TalkingCat()

salem.meow() # Méthode héritée de Cat

salem.say("Hello!") # Méthode héritée de Talker

✅ TalkingCat **a accès aux méthodes des deux parents (Cat et Talker)**.

**3️⃣ Résolution des Conflits d’Héritage**

* **Problème du diamant** 🛑 : Si plusieurs classes définissent la même méthode, laquelle est exécutée ?
* **Python suit le MRO (Method Resolution Order)** :

python

CopierModifier

print(TalkingCat.\_\_mro\_\_) # Affiche l'ordre de résolution des méthodes

**✏️ Exercice : Hiérarchie et Héritage Multiple**

Nous allons :

1. **Créer une classe Image (parent de toutes les images).**
2. **Créer des sous-classes GIF et JPG.**
3. **Surcharger display() pour afficher le type d’image.**

**📌 Implémentation en Python**

python

CopierModifier

class File:

"""Représente un fichier générique."""

def \_\_init\_\_(self, name, size):

self.name = name

self.size = size

def display(self):

"""Affiche les informations du fichier."""

print(f"Fichier : {self.name} ({self.size} Mo)")

class Image(File):

"""Classe parent pour toutes les images."""

def \_\_init\_\_(self, name, size, resolution):

super().\_\_init\_\_(name, size)

self.resolution = resolution

def display(self):

"""Affiche le fichier image."""

print(f"📷 Image : {self.name}, résolution : {self.resolution}")

class GIF(Image):

"""Sous-classe de Image pour les fichiers GIF."""

def \_\_init\_\_(self, name, size, resolution, frame\_rate):

super().\_\_init\_\_(name, size, resolution)

self.frame\_rate = frame\_rate

def display(self):

"""Affiche un message spécial pour les GIFs."""

print(f"📹 GIF : {self.name}, {self.resolution}, {self.frame\_rate} fps")

class JPG(Image):

"""Sous-classe de Image pour les fichiers JPG."""

def \_\_init\_\_(self, name, size, resolution, quality):

super().\_\_init\_\_(name, size, resolution)

self.quality = quality

def display(self):

"""Affiche un message spécial pour les JPG."""

print(f"🖼 JPG : {self.name}, {self.resolution}, Qualité : {self.quality}")

**🚀 Tests**

python

CopierModifier

gif = GIF("animation.gif", 2.0, "800x600", 30)

jpg = JPG("photo.jpg", 3.5, "1920x1080", 95)

gif.display()

jpg.display()

✅ **Résultat attendu :**

yaml

CopierModifier

📹 GIF : animation.gif, 800x600, 30 fps

🖼 JPG : photo.jpg, 1920x1080, Qualité : 95

Solution openclassrooms :  
  
"""Définit les classes propres à notre forum. ;)"""

from abc import ABC

class File(ABC):

"""Fichier."""

def \_\_init\_\_(self, name, size):

"""Initialise le nom et la taille."""

self.name = name

self.size = size

def display(self):

"""Affiche le fichier."""

pass

class ImageFile(File):

"""Fichier image."""

def display(self):

"""Affiche l'image."""

print(f"Fichier image '{self.name}'.")

class GifImageFile(ImageFile):

"""Fichier image Gif."""

def display(self):

"""Affiche l'image."""

super().display()

print("L'image est de type 'Gif'.")

class PNGImageFile(ImageFile):

"""Fichier image PNG."""

def display(self):

"""Affiche l'image."""

super().display()

print("L'image est de type 'PNG'.")

class User:

"""Utilisateur."""

def \_\_init\_\_(self, username, password):

"""Initialise le nom d'utilisateur et le mot de passe."""

self.username = username

self.password = password

def login(self):

"""Connecte l'utilisateur."""

print(f"L'utilisateur {self.username} est connecté.")

def post(self, thread, content, file=None):

"""Poste un message dans un fil de discussion."""

if file:

post = FilePost(self, "aujourd'hui", content, file)

else:

post = Post(user=self, time\_posted="aujourd'hui", content=content)

thread.add\_post(post)

return post

def make\_thread(self, title, content):

"""Créé un nouveau fil de discussion."""

post = Post(self, "aujourd'hui", content)

return Thread(title, "aujourd'hui", post)

def \_\_str\_\_(self):

"""représentation de l'utilisateur."""

return self.username

class Moderator(User):

"""Utilisateur modérateur."""

def edit(self, post, content):

"""Modifie un message."""

post.content = content

def delete(self, thread, post):

"""Supprime un message."""

index = thread.posts.index(post)

del thread.posts[index]

class Post:

"""Message."""

def \_\_init\_\_(self, user, time\_posted, content):

"""Initialise l'utilisateur, la date et le contenu."""

self.user = user

self.time\_posted = time\_posted

self.content = content

def display(self):

"""Affiche le message."""

print(f"Message posté par {self.user} le {self.time\_posted}:")

print(self.content)

class FilePost(Post):

"""Message comportant un fichier."""

def \_\_init\_\_(self, user, time\_posted, content, file):

"""Initialise le fichier."""

super().\_\_init\_\_(user, time\_posted, content)

self.file = file

def display(self):

"""Affiche le contenu et le fichier."""

super().display()

print("pièce jointe:")

self.file.display()

class Thread:

"""Fil de discussions."""

def \_\_init\_\_(self, title, time\_posted, post):

"""Initialise le titre, la date et les posts.

Attention ici: on commence par un seul post, celui du sujet.

Les réponses à ce post ne pourrons s'ajouter qu'ultérieurement.

En effet, on ne créé pas directement un nouveau fil avec des réponses. ;)

"""

self.title = title

self.time\_posted = time\_posted

self.posts = [post]

def display(self):

"""Affiche le fil de discussion."""

print("----- THREAD -----")

print(f"titre: {self.title}, date: {self.time\_posted}")

print()

for post in self.posts:

post.display()

print()

print("------------------")

def add\_post(self, post):

"""Ajoute un post."""

self.posts.append(post)

**Résumé et Points Clés : Utilisation des Objets dans des Collections**

**1️⃣ Stocker des Objets dans des Collections**

* **Les listes (list)** permettent de stocker plusieurs objets dans un ordre spécifique.
* **Les dictionnaires (dict)** associent des clés (immuables) à des objets.

📌 **Exemple : Liste d’objets Person**

python

CopierModifier

class Person:

def \_\_init\_\_(self, name):

self.name = name

def walk(self):

print(f"{self.name} marche.")

volunteers = [Person("Alice"), Person("Bob"), Person("Carol")]

for volunteer in volunteers:

volunteer.walk()

✅ **Chaque objet Person peut être parcouru et utilisé dans une boucle.**

📌 **Exemple : Dictionnaire d’objets outils**

python

CopierModifier

toolbox = {

"screwdriver": Screwdriver(50),

"hammer": Hammer(),

"nails": [Nail(), Nail(), Nail()],

}

✅ **On peut stocker des objets dans un dictionnaire pour y accéder par clé.**

**2️⃣ Duck Typing en Python**

* **Python ne vérifie pas le type** d’un objet, il vérifie juste s’il possède l’attribut ou la méthode attendue.
* **Problème potentiel :** Si un objet ne possède pas une méthode utilisée dans la collection, le programme plante.

📌 **Exemple de bug potentiel**

python

CopierModifier

volunteers = [Person("Alice"), Fish("Wanda"), Person("Bob")]

for volunteer in volunteers:

volunteer.walk() # Erreur : Fish n'a pas la méthode walk()

❌ AttributeError: 'Fish' object has no attribute 'walk'

📌 **Solution : Vérifier avec hasattr()**

python

CopierModifier

for volunteer in volunteers:

if hasattr(volunteer, "walk"):

volunteer.walk()

✅ **Évite l’erreur en s’assurant que l’objet possède walk().**

**3️⃣ Stratégies pour Gérer les Erreurs**

| **Méthode** | **Explication** |
| --- | --- |
| **Documentation (docstrings)** | Explique le comportement attendu des objets. |
| **Typage (typing)** | Utiliser List[Type] pour indiquer quel type est attendu. |
| **Programmation défensive (if checks)** | Vérifier si l’objet possède l’attribut/méthode. |
| **Exceptions (try/except)** | Capturer et gérer les erreurs d’exécution. |

📌 **Exemple : Typage avec typing**

python

CopierModifier

from typing import List

def highest(numbers: List[int]) -> int:

return max(numbers)

✅ **Indique que numbers doit être une List[int], mais Python ne l’applique pas strictement.**

**✏️ Exercice : Manipuler un Fil de Discussion**

1. **Créer un utilisateur et un modérateur.**
2. **L’utilisateur crée un fil de discussion et y ajoute un message.**
3. **Le modérateur répond.**
4. **L’utilisateur poste un message hors sujet.**
5. **Le modérateur supprime ce message.**
6. **L’utilisateur poste un message avec une image.**
7. **Afficher l’état du fil après chaque action.**

**📌 Implémentation en Python**

python

CopierModifier

# Création des utilisateurs

user = User("Alice", "password123")

moderator = Moderator("Bob", "securepass")

# L’utilisateur crée un fil de discussion

thread = user.make\_thread("Recettes de gâteaux", "J’adore le chocolat !")

# Le modérateur répond

moderator.post(thread, "Moi aussi ! Voici ma recette préférée.")

# L’utilisateur poste un message hors sujet

user.post(thread, "Quelqu'un a vu le dernier film de science-fiction ?")

# Affichage avant suppression

print("📌 État du fil avant suppression :")

for post in thread.posts:

post.display()

# Le modérateur supprime le message hors sujet

moderator.delete(thread, thread.posts[-1])

# L’utilisateur ajoute un post avec une image

image = Image("gateau.jpg", 2.5, "1920x1080")

file\_post = FilePost(user, "Regardez ce gâteau !", image)

thread.add\_post(file\_post)

# Affichage final

print("\n📌 État final du fil :")

for post in thread.posts:

post.display()

✅ **Sortie attendue :**

bash

CopierModifier

📌 État du fil avant suppression :

Alice a écrit : J’adore le chocolat !

Bob a écrit : Moi aussi ! Voici ma recette préférée.

Alice a écrit : Quelqu'un a vu le dernier film de science-fiction ?

📌 État final du fil :

Alice a écrit : J’adore le chocolat !

Bob a écrit : Moi aussi ! Voici ma recette préférée.

📎 Fichier attaché : gateau.jpg

Alice a écrit : Regardez ce gâteau !

solution openclassrooms :  
  
"""Définit les classes propres à notre forum. ;)"""

import time

from abc import ABC

class File(ABC):

"""Fichier."""

def \_\_init\_\_(self, name, size):

"""Initialise le nom et la taille."""

self.name = name

self.size = size

def display(self):

"""Affiche le fichier."""

pass

class ImageFile(File):

"""Fichier image."""

def display(self):

"""Affiche l'image."""

print(f"Fichier image '{self.name}'.")

class GifImageFile(ImageFile):

"""Fichier image Gif."""

def display(self):

"""Affiche l'image."""

super().display()

print("L'image est de type 'Gif'.")

class PNGImageFile(ImageFile):

"""Fichier image PNG."""

def display(self):

"""Affiche l'image."""

super().display()

print("L'image est de type 'PNG'.")

class User:

"""Utilisateur."""

def \_\_init\_\_(self, username, password):

"""Initialise le nom d'utilisateur et le mot de passe."""

self.username = username

self.password = password

def login(self):

"""Connecte l'utilisateur."""

print(f"L'utilisateur {self.username} est connecté.")

def post(self, thread, content, file=None):

"""Poste un message dans un fil de discussion."""

if file:

post = FilePost(self, "aujourd'hui", content, file)

else:

post = Post(user=self, time\_posted="aujourd'hui", content=content)

thread.add\_post(post)

return post

def make\_thread(self, title, content):

"""Créé un nouveau fil de discussion."""

post = Post(self, "aujourd'hui", content)

return Thread(title, "aujourd'hui", post)

def \_\_str\_\_(self):

"""représentation de l'utilisateur."""

return self.username

class Moderator(User):

"""Utilisateur modérateur."""

def edit(self, post, content):

"""Modifie un message."""

post.content = content

def delete(self, thread, post):

"""Supprime un message."""

index = thread.posts.index(post)

del thread.posts[index]

class Post:

"""Message."""

def \_\_init\_\_(self, user, time\_posted, content):

"""Initialise l'utilisateur, la date et le contenu."""

self.user = user

self.time\_posted = time\_posted

self.content = content

def display(self):

"""Affiche le message."""

print(f"-- Message posté par {self.user} {self.time\_posted} --")

print(self.content)

class FilePost(Post):

"""Message comportant un fichier."""

def \_\_init\_\_(self, user, time\_posted, content, file):

"""Initialise le fichier."""

super().\_\_init\_\_(user, time\_posted, content)

self.file = file

def display(self):

"""Affiche le contenu et le fichier."""

super().display()

print("pièce jointe:")

self.file.display()

class Thread:

"""Fil de discussions."""

def \_\_init\_\_(self, title, time\_posted, post):

"""Initialise le titre, la date et les posts.

Attention ici: on commence par un seul post, celui du sujet.

Les réponses à ce post ne pourrons s'ajouter qu'ultérieurement.

En effet, on ne créé pas directement un nouveau fil avec des réponses. ;)

"""

self.title = title

self.time\_posted = time\_posted

self.posts = [post]

def display(self):

"""Affiche le fil de discussion."""

print("----- THREAD -----")

print(f"titre: {self.title}, date: {self.time\_posted}")

print()

for post in self.posts:

post.display()

print()

print("------------------")

def add\_post(self, post):

"""Ajoute un post."""

self.posts.append(post)

def main():

"""Lance le code principal."""

user = User("John", "superpassword")

moderator = Moderator("Lucie", "helloworld")

cake\_thread = user.make\_thread("Gâteau à la vanille 🍰 ???", "Vous aimez ou non ?")

cake\_thread.display()

moderator.post(cake\_thread, content="Oui j'aime beaucoup ! 😚")

cake\_thread.display()

irrelevant\_post = user.post(cake\_thread, content="Et vous aimez les voitures ?")

response = moderator.post(cake\_thread, content="C'est hors sujet sur ce forum 😕")

cake\_thread.display()

print()

print("après quelques minutes, le modérateur supprime les messages hors sujets...")

print()

# importer time n'était pas necessaire, c'est un plus:

time.sleep(2)

moderator.delete(cake\_thread, irrelevant\_post)

moderator.delete(cake\_thread, response)

cake\_thread.display()

image = PNGImageFile(name="image de gâteau", size=3)

user.post(cake\_thread, content="Voici une image de mon gâteau !", file=image)

moderator.post(cake\_thread, "Woah, sublime !")

cake\_thread.display()

main()

QUIZZ :  
  
**Quiz : Utiliser le Comportement Hérité en Python**

**✅ Question 1**

**Quand une classe hérite d’une autre classe, elle acquiert :**  
✅ **Ses méthodes**  
✅ **Ses variables**  
❌ Son nom (Le nom d'une classe enfant est distinct de celui du parent.)

**Explication :** Une sous-classe hérite des méthodes et des variables de la classe parent, mais elle a son propre nom.

**✅ Question 2**

**Une classe de base abstraite :**  
✅ **Ne peut pas être instanciée, mais on peut en hériter.**

**Explication :** Une **classe abstraite** sert uniquement de modèle pour ses sous-classes et ne peut pas être instanciée directement.

**✅ Question 3**

**Le duck typing, c’est :**  
✅ **Lorsque Python s’intéresse à ce qu’un objet peut faire, et non à son type.**

**Explication :** En Python, on vérifie **les méthodes et attributs** d’un objet plutôt que son type explicite.

**✅ Question 4**

python

CopierModifier

class A:

def func(self):

pass

class B:

def func(self, number):

pass

elements = [A(), B()]

for item in elements:

item.func()

❌ **Il plantera parce qu’on ne peut pas invoquer func sans argument en B.**

**Explication :** B.func(self, number) attend un argument supplémentaire (number), donc item.func() sans argument provoquera une erreur.

**✅ Question 5**

**Raisons d’utiliser l’héritage :**  
✅ **Il vous permet de représenter la relation entre des concepts.**  
✅ **Il réduit le besoin de dupliquer du code.**  
✅ **Pour décomposer du code en segments logiques, pour qu’il soit plus facile à développer en équipe.**  
❌ **Pour garantir que les sous-classes fassent toujours les mêmes choses que leurs classes parents.** (*Une sous-classe peut modifier le comportement hérité.*)

**✅ Question 6**

**Ligne correcte pour appeler le constructeur d’une classe parent :**  
✅ **super().\_\_init\_\_()**  
✅ **ParentClass.\_\_init\_\_(self)**

**Explication :**

* super().\_\_init\_\_() est la méthode recommandée pour appeler le constructeur parent.
* ParentClass.\_\_init\_\_(self) est une alternative fonctionnelle, mais **moins flexible**.

**✅ Question 7**

**Si une classe a deux classes parents qui définissent la même méthode, alors Python…**  
✅ **Sélectionnera la définition de méthode de la classe la plus à gauche dans la définition de classe.**

**Explication :** Python suit l’ordre MRO (**Method Resolution Order**) et cherche d’abord dans le premier parent défini à gauche.

**✅ Question 8**

**Parmi ces propositions, lesquelles sont vraies ?**  
✅ **Un objet mutable est un objet dont on peut modifier l’état.**  
✅ **Un objet peut être mutable ou immutable.**

**Explication :**

* **Mutable** : Listes, dictionnaires, objets modifiables.
* **Immutable** : Entiers (int), chaînes (str), tuples (tuple).

**🎯 Résumé des Réponses**

| **Question** | **Réponse Correcte** |
| --- | --- |
| **1** | Ses méthodes, ses variables |
| **2** | Ne peut pas être instanciée, mais on peut en hériter |
| **3** | Python s’intéresse à ce qu’un objet peut faire, et non à son type |
| **4** | Il plantera parce qu’on ne peut pas invoquer func sans argument en B |
| **5** | Représente la relation entre concepts, réduit le code, facilite le développement en équipe |
| **6** | super().\_\_init\_\_(), ParentClass.\_\_init\_\_(self) |
| **7** | Sélectionne la méthode de la classe la plus à gauche |
| **8** | Un objet mutable peut être modifié, un objet peut être mutable ou immutable |

**Résumé et Points Clés : Structurer son Code avec des Modules et des Classes**

**1️⃣ Pourquoi Structurer son Code ?**

* **Facilite la maintenance et l’extension** : On peut modifier ou ajouter du code sans tout casser.
* **Améliore la lisibilité** : On comprend mieux le rôle de chaque partie du programme.
* **Permet de travailler en équipe** : Chacun peut gérer une partie spécifique du code.

**2️⃣ Principes Clés pour un Code Structuré**

✅ **Principe de découvrabilité** :  
👉 Organiser les fichiers et dossiers pour **retrouver facilement** une fonctionnalité.

✅ **Principe de simplicité** :  
👉 Chaque fonction/classe ne doit faire **qu’une seule chose bien définie**.

✅ **Principe de style** :  
👉 Utiliser **une convention de nommage cohérente** (ex : camelCase, snake\_case).

**3️⃣ Fichiers, Modules et Packages en Python**

| **Terme** | **Définition** |
| --- | --- |
| **Module** | Un **fichier Python** (.py) contenant du code réutilisable. |
| **Package** | Un **dossier** contenant plusieurs modules et un \_\_init\_\_.py. |
| **Import** | Permet d’utiliser du code d’un autre fichier/module. |

📌 **Exemple : Structure d’un Projet**

bash

CopierModifier

projet\_forum/

│── main.py # Point d'entrée du programme

│── models/ # Package contenant les modèles de données

│ │── \_\_init\_\_.py

│ │── user.py # Classe User et Moderator

│ │── post.py # Classe Post et FilePost

│ │── thread.py # Classe Thread

│── utils/ # Package contenant des fonctions utilitaires

│ │── \_\_init\_\_.py

│ │── helpers.py # Fonctions d’aide générales

📌 **Exemple d’import de module**

python

CopierModifier

# Importer une classe depuis un module

from models.user import User, Moderator

alice = User("Alice", "password123")

📌 **Exemple d’un \_\_init\_\_.py dans un package**

python

CopierModifier

# models/\_\_init\_\_.py

\_\_all\_\_ = ["user", "post", "thread"] # Permet d’importer tout ce qui est listé ici

**4️⃣ L’Instruction if \_\_name\_\_ == "\_\_main\_\_"**

📌 **Utilisée pour exécuter un script uniquement lorsqu’il est le point d’entrée du programme.**

python

CopierModifier

if \_\_name\_\_ == "\_\_main\_\_":

print("Ceci s'exécute uniquement si ce fichier est lancé directement.")

✅ **Utile pour éviter l’exécution de code quand on importe un module.**

**✏️ Exercice : Réorganiser un Programme**

**Objectif : Transformer un fichier monolithique en projet bien structuré.**

1. **Créer un package models/ pour les classes (User, Post, Thread).**
2. **Créer un package utils/ pour les fonctions d’aide.**
3. **Mettre le point d’entrée du programme dans main.py.**
4. **Utiliser import pour relier les fichiers entre eux.**

📌 **Exemple de main.py après refactorisation**

python

CopierModifier

from models.user import User, Moderator

from models.post import Post, FilePost

from models.thread import Thread

if \_\_name\_\_ == "\_\_main\_\_":

alice = User("Alice", "password123")

mod = Moderator("Bob", "securepass")

thread = alice.make\_thread("Recettes de gâteaux", "Qui aime les tartes ?")

mod.post(thread, "Moi j'adore la tarte aux fraises !")

for post in thread.posts:

post.display()

**Résumé et Points Clés : Décomposer un Problème en Programmation Orientée Objet**

**1️⃣ Pourquoi Décomposer un Problème ?**

* **Facilite la compréhension** : On divise un problème complexe en plusieurs parties simples.
* **Améliore la maintenabilité** : Un code bien structuré est plus facile à modifier et étendre.
* **Permet de réutiliser le code** : On évite la duplication grâce aux classes et modules.

**2️⃣ Étapes de Décomposition**

✅ **1. Définir les fonctionnalités**

* Que doit faire le programme ?
* Comment garantir son extensibilité pour de futures fonctionnalités ?

✅ **2. Identifier les objets et leurs relations**

* Quels objets existent ?
* **Relations "is-a" (héritage)** → ex : FilmVHS est un Film.
* **Relations "has-a" (composition)** → ex : Un Ami a un Film emprunté.

✅ **3. Concevoir les interfaces**

* Comment les objets interagissent-ils entre eux ?
* Quels attributs et méthodes doivent-ils avoir ?

**3️⃣ 📌 Exercice : Organiser une Bibliothèque de Films**

**📌 Analyse des données**

| **Élément** | **Détails** |
| --- | --- |
| **Un film** | Nom, date, type (VHS ou DVD), lieu (chez moi ou chez un ami). |
| **Types de films** | VHS et DVD (héritage de Film). |
| **Gestion des films** | Trier par date, nom, type, récupérer un film au hasard. |
| **Amis** | Nom, film emprunté (optionnel). |
| **Gestion des prêts** | Lister les films prêtés, savoir qui possède un film. |

**4️⃣ 📌 Implémentation en Python**

📌 **1️⃣ Création des classes**

python

CopierModifier

import random

class Film:

"""Représente un film générique."""

def \_\_init\_\_(self, title, year, film\_type):

self.title = title

self.year = year

self.film\_type = film\_type.lower()

self.location = "chez moi" # Par défaut, les films sont chez moi.

def \_\_str\_\_(self):

return f"{self.title} ({self.year}) - {self.film\_type.capitalize()} - {self.location}"

class FilmVHS(Film):

"""Film VHS."""

def \_\_init\_\_(self, title, year):

super().\_\_init\_\_(title, year, "VHS")

class FilmDVD(Film):

"""Film DVD."""

def \_\_init\_\_(self, title, year):

super().\_\_init\_\_(title, year, "DVD")

class Ami:

"""Représente un ami qui peut emprunter un film."""

def \_\_init\_\_(self, name, borrowed\_film=None):

self.name = name

self.borrowed\_film = borrowed\_film # Film emprunté (optionnel)

def \_\_str\_\_(self):

return f"{self.name} a emprunté {self.borrowed\_film.title}" if self.borrowed\_film else f"{self.name} n'a rien emprunté."

class Bibliothèque:

"""Gère la collection de films et les emprunts."""

def \_\_init\_\_(self):

self.films = []

self.amis = []

def ajouter\_film(self, film):

"""Ajoute un film à la bibliothèque."""

self.films.append(film)

def trier\_films(self, critère="nom"):

"""Trie les films selon le critère donné."""

if critère == "nom":

self.films.sort(key=lambda film: film.title)

elif critère == "date":

self.films.sort(key=lambda film: film.year)

elif critère == "type":

self.films.sort(key=lambda film: film.film\_type)

def film\_au\_hasard(self):

"""Retourne un film aléatoire."""

return random.choice(self.films) if self.films else None

def preter\_film(self, film, ami):

"""Prête un film à un ami."""

if film in self.films and film.location == "chez moi":

film.location = f"chez {ami.name}"

ami.borrowed\_film = film

else:

print(f"Le film {film.title} n'est pas disponible.")

def films\_prêtés(self):

"""Retourne la liste des films prêtés."""

return [film for film in self.films if "chez" in film.location]

def qui\_a\_ce\_film(self, title):

"""Retourne qui possède un film donné."""

for ami in self.amis:

if ami.borrowed\_film and ami.borrowed\_film.title == title:

return f"{title} est chez {ami.name}"

return f"{title} est chez moi."

**5️⃣ 📌 Nettoyage et Importation des Données**

📌 **2️⃣ Nettoyage des données**

python

CopierModifier

films\_data = [

("Blade Runner (1982)", "vhf"),

("Alien : Le 8ème Passager (1979)", "vhf"),

("2001 : L'Odyssée de l'espace (1968)", "VhF"),

("Matrix (1999)", "DVD"),

("Interstellar (2014)", "dvD"),

("L'Empire contre-attaque (1980)", "vhf"),

("Retour vers le futur (1985)", "vhf"),

("La Guerre des Étoiles (1977)", "vhf"),

("L'Armée des 12 singes (1995)", "dVd"),

("Terminator 2 : Le Jugement dernier (1991)", "DVD"),

]

friends\_data = [

("Paul", "Blade Runner"),

("Lucie",),

("Zoé", "Terminator 2 : Le Jugement dernier"),

]

# Création de la bibliothèque

bibliothèque = Bibliothèque()

# Ajout des films après nettoyage des données

for title, film\_type in films\_data:

clean\_title, year = title.rsplit(" (", 1) # Sépare le titre et l'année

year = int(year.strip(")")) # Convertit en int

film\_type = film\_type.lower().strip() # Normalisation du type

film = FilmVHS(clean\_title, year) if film\_type == "vhf" else FilmDVD(clean\_title, year)

bibliothèque.ajouter\_film(film)

# Ajout des amis et assignation des prêts

for friend\_data in friends\_data:

name = friend\_data[0]

borrowed\_title = friend\_data[1] if len(friend\_data) > 1 else None

ami = Ami(name)

bibliothèque.amis.append(ami)

if borrowed\_title:

for film in bibliothèque.films:

if borrowed\_title in film.title:

bibliothèque.preter\_film(film, ami)

**6️⃣ 📌 Tests**

📌 **3️⃣ Tester le système**

python

CopierModifier

# Affichage de tous les films triés par nom

bibliothèque.trier\_films("nom")

for film in bibliothèque.films:

print(film)

# Vérification des films prêtés

print("\n🎥 Films prêtés :")

for film in bibliothèque.films\_prêtés():

print(film)

# Qui possède un film ?

print("\n🔎 Où est 'Blade Runner' ?")

print(bibliothèque.qui\_a\_ce\_film("Blade Runner"))

✅ **Sortie attendue :**

java

CopierModifier

Blade Runner (1982) - VHS - chez Paul

...

🎥 Films prêtés :

Blade Runner (1982) - VHS - chez Paul

Terminator 2 : Le Jugement dernier (1991) - DVD - chez Zoé

🔎 Où est 'Blade Runner' ?

Blade Runner est chez Paul.

**Résumé et Points Clés : Gestion des Exceptions en Python**

**1️⃣ Qu’est-ce qu’une Exception ?**

Une **exception** est un message d'erreur généré lorsqu’un problème survient pendant l’exécution d’un programme.

📌 **Exemples courants d’exceptions en Python** :

| **Exception** | **Cause** |
| --- | --- |
| ZeroDivisionError | Division par zéro (1/0) |
| NameError | Variable non définie |
| IndexError | Index hors de portée dans une liste |
| KeyError | Clé non trouvée dans un dictionnaire |
| TypeError | Opération incompatible avec le type ("3" + 3) |

**2️⃣ Lever (raise) une Exception**

On peut déclencher une exception avec raise :

python

CopierModifier

def get\_half\_even\_number(number):

if number % 2 == 0:

return number / 2

else:

raise ValueError(f"Seuls les nombres pairs sont acceptés. Reçu : {number}")

✅ **Si number est impair, une erreur est levée avec un message explicatif.**

**3️⃣ Gérer une Exception avec try-except**

python

CopierModifier

def increase\_percent(initial\_value, after\_value):

try:

return (after\_value / initial\_value) \* 100

except ZeroDivisionError:

return 0 # Empêche une erreur si `initial\_value` est 0

except Exception as error:

print(f"Erreur inattendue : {error}")

raise # Relance l'erreur pour ne pas masquer le problème

✅ **Gestion fine des erreurs :**

* ZeroDivisionError est traité avec une valeur par défaut.
* Toute autre erreur est affichée et relancée.

**4️⃣ Créer une Exception Personnalisée**

Une exception est un **objet Python** qui hérite de Exception :

python

CopierModifier

class InvalidAddressException(Exception):

"""Exception pour une adresse invalide."""

def \_\_init\_\_(self, address):

super().\_\_init\_\_(f"Adresse invalide : {address}")

📌 **Utilisation dans une classe :**

python

CopierModifier

class OwlContactSystem:

def \_\_init\_\_(self, address):

if not address.endswith("@hogwarts.com"):

raise InvalidAddressException(address)

self.address = address

✅ **Si l’adresse est incorrecte, l’erreur InvalidAddressException est levée.**

**5️⃣ Hiérarchie des Exceptions Personnalisées**

Créer une **classe parent** pour toutes les exceptions d’une application :

python

CopierModifier

class MyAppException(Exception):

"""Exception de base pour l'application."""

pass

class UsernameTooShortException(MyAppException):

"""Le nom d'utilisateur est trop court."""

pass

class WeakPasswordException(MyAppException):

"""Le mot de passe est trop faible."""

pass

✅ **Avantage : On peut attraper toutes les erreurs spécifiques de l’application en capturant MyAppException.**

**✏️ Exercice : Vérifier un Nom d’Utilisateur et un Mot de Passe**

1. **Créer des exceptions personnalisées pour :**
   * Nom d’utilisateur trop court.
   * Mot de passe insuffisant (au moins une lettre et un chiffre).
2. **Modifier la classe User pour lever ces erreurs.**
3. **Tester avec des entrées valides et invalides.**

📌 **Implémentation**

python

CopierModifier

import re

class UserException(Exception):

"""Exception de base pour les erreurs utilisateur."""

pass

class UsernameTooShortException(UserException):

"""Erreur si le nom d'utilisateur a moins de 3 caractères."""

def \_\_init\_\_(self, username):

super().\_\_init\_\_(f"Nom d'utilisateur trop court : {username}")

class WeakPasswordException(UserException):

"""Erreur si le mot de passe est trop faible."""

def \_\_init\_\_(self):

super().\_\_init\_\_("Le mot de passe doit contenir au moins une lettre et un chiffre.")

class User:

"""Représente un utilisateur avec un nom et un mot de passe sécurisé."""

def \_\_init\_\_(self, username, password):

if len(username) < 3:

raise UsernameTooShortException(username)

if not re.search(r'[A-Za-z]', password) or not re.search(r'\d', password):

raise WeakPasswordException()

self.username = username

self.password = password

def \_\_str\_\_(self):

return f"Utilisateur : {self.username}"

**🚀 Test du Code**

python

CopierModifier

try:

user1 = User("Al", "pass123") # Trop court

except UserException as e:

print(f"Erreur : {e}")

try:

user2 = User("Alice", "abcdef") # Pas de chiffre

except UserException as e:

print(f"Erreur : {e}")

try:

user3 = User("Bob", "password1") # Valide

print(user3)

except UserException as e:

print(f"Erreur : {e}")

✅ **Sortie attendue :**

yaml

CopierModifier

Erreur : Nom d'utilisateur trop court : Al

Erreur : Le mot de passe doit contenir au moins une lettre et un chiffre.

Utilisateur : Bob

QUIZZ !  
  
**Revenez sur vos acquis**

Félicitations : vous êtes parvenu à la fin du cours ! Si vous débutez en programmation orientée objet, la meilleure manière de vous améliorer dans toutes les compétences liées à la programmation est de les mettre en pratique ! Concevez des systèmes, construisez-les, et apprenez de vos erreurs.

Gardez cela en tête pendant que nous récapitulons ce que nous avons appris dans ce cours...

**Les classes et les objets**

La **programmation orientée objet** est un paradigme de programmation courant, dans lequel des **objets** – créés à partir de plans que l’on appelle des **classes** – interagissent pour produire des comportements.

Les **objets**ont un **état**et un **comportement**. L’état désigne les **variables**et les **données**qui correspondent à cet objet, alors que le comportement se trouve dans les **méthodes**de cet objet.

**L'héritage**

Les classes et les objets peuvent avoir différentes relations entre eux, qui représentent différents concepts. Les relations **d’héritage**(« is-a », ou « est-un·e ») permettent aux classes enfants d’hériter de fonctionnalités de leurs classes parents et de les surcharger, tandis que les relations de **composition**(« has-a » ou « a-un·e ») concernent un objet contenant une référence à un ou des objet(s) autre(s). Ces dernières sont fréquemment utilisées lorsqu’un objet contient ou est composé d’autres objets.

**La décomposition**

À mesure qu’un problème de programmation devient plus difficile, il en va de même pour le travail de conception nécessaire. Nous savons que nous pouvons **décomposer un problème** en parties plus petites qui interagissent, et découper notre programme en **modules**et en **paquets**.

**Les exceptions**

Enfin, nous avons découvert – en acquérant des connaissances sur les **exceptions**et par un apprentissage de première main – que les choses peuvent mal se passer au sein des programmes. Il existe des**exceptions intégrées**qui nous avertissent quand nous avons fait quelque chose qui ne fonctionne pas, et des **exceptions personnalisées**que nous pouvons écrire nous-mêmes, pour qu’elles soient **déclenchées**en cas de problème.

Si tout cela vous semble familier, c’est formidable ! Sinon, il serait peut-être judicieux de jeter un coup d’œil aux chapitres correspondants avant de passer à la suite. En tous les cas, nous vous souhaitons une excellente continuation dans vos projets de développement et votre apprentissage continu de la programmation avec Python. 🎓