Concepts Python utilisés dans CinéMax

Guide des concepts Python les plus intéressants du projet avec exemples de code.

Sommaire par Niveau

Avancé

Intermédiaire

Fondamental

Concepts Avancés

1. **Décorateurs (@)**

Les décorateurs modifient le comportement des fonctions.

```
# Décorateur Flask pour créer une route
@app.route("/api/movies", methods=["GET"])
def get_movies():
    movies = get_movies_from_db()
    return jsonify(movies)

# Décorateur personnalisé pour vérifier l'authentification
def require_login(func):
    def wrapper(*args, **kwargs):
        if "user_id" not in session:
            return jsonify({"error": "Non authentifié"}), 401
        return func(*args, **kwargs)
    return wrapper

@app.route("/api/bookings")
@require_login
def my_bookings():
    # Automatiquement vérifiée que l'utilisateur est connecté
    return jsonify({"bookings": []})
```

Cas d'usage dans CinéMax :

- `@app.route()` : Définit les URLs accessibles
- Protéger les routes nécessitant une authentification
- Mesurer les performances des fonctions

2. **Context Managers (with)**

Gère automatiquement l'allocation et la libération de ressources.

```
# SQLite avec context manager
with sqlite3.connect(DB_PATH) as conn:
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM movies")
    movies = cursor.fetchall()
# La connexion est automatiquement fermée

# Context manager personnalisé
class DatabaseConnection:
    def __init__(self, db_path):
        self.db_path = db_path
        self.conn = None

    def __enter__(self):
        self.conn = sqlite3.connect(self.db_path)
        return self.conn

    def __exit__(self, exc_type, exc_val, exc_tb):
        if self.conn:
            self.conn.close()

# Utilisation
with DatabaseConnection("cinema.db") as conn:
    # La connexion se ferme automatiquement
    pass
```

3. **Générateurs (yield)**

Crée des fonctions qui retournent des valeurs une par une (économise la mémoire).

```
# Générateur simple
def get_available_dates(days=5):
    today = datetime.now()
    for d in range(days):
        dt = today + timedelta(days=d)
        yield {
            "iso": dt.strftime("%Y-%m-%d"),
            "label": dt.strftime("%a %d %b"),
        }

# Utilisation
for date in get_available_dates(10):  # Ne crée pas une liste de 10 items en mémoire
    print(date)

# Expression générateur (utile avec de grands datasets)
movie_titles = (m["title"] for m in MOVIES)  # Pas stocké en mémoire
```

Avantage : Consomme beaucoup moins de mémoire pour de grandes listes.

4. **Closures**

Une fonction qui retourne une autre fonction capable d'accéder aux variables parentes.

```
def make_price_calculator(base_price):
    """Crée une fonction qui retient base_price"""
    def calculate_with_discount(quantity):
        if quantity >= 5:
            return base_price * quantity * 0.9  # 10% de réduction
        return base_price * quantity
    return calculate_with_discount

# Utilisation dans CinéMax
student_prices = make_price_calculator(8.00)  # Prix réduit
adult_prices = make_price_calculator(12.50)   # Prix normal

print(student_prices(3))  # 24.00
print(student_prices(5))  # 36.00 (avec réduction 10%)
print(adult_prices(5))    # 62.50
```

Cas d'usage : Créer des calculatrices de prix personnalisées, des fonctions de configuration.

5. **Type Hints (Annotations de type)**

Indique quel type les paramètres et retour devraient avoir. Aide à détecter les erreurs.

```
from typing import List, Dict, Optional, Tuple

def create_user(email: str, password: str) -> Dict[str, any]:
    """Crée un utilisateur"""
    return {
        "email": email,
        "password": generate_password_hash(password)
    }

def get_user_bookings(user_id: int) -> List[Dict]:
    """Retourne les réservations"""
    pass

def find_user(user_id: int) -> Optional[Dict]:
    """Retourne None si pas trouvé"""
    pass

def parse_date(date: str) -> Tuple[str, str]:
    """Retourne (date_iso, date_lisible)"""
    pass

# Avec mypy ou Pylance, ça détecte les erreurs:
bookings = get_user_bookings("123")  # □ Erreur: "123" n'est pas un int
```

6. **Asynchrone (async/await)**

Exécute plusieurs tâches en parallèle sans bloquer.

```
import asyncio

async def fetch_movie(movie_id):
    """Fonction asynchrone (simule une requête lente)"""
    await asyncio.sleep(1)  # Attend 1 seconde
    return {"id": movie_id, "title": "Dune"}

async def main():
    # Fetch 3 films en parallèle (1 seconde au total, pas 3)
    results = await asyncio.gather(
        fetch_movie(1),
        fetch_movie(2),
        fetch_movie(3)
    )
    print(results)
    # Output: [{'id': 1, ...}, {'id': 2, ...}, {'id': 3, ...}]

# Exécuter
# asyncio.run(main())
```

Cas d'usage dans CinéMax : Charger plusieurs films ou réservations en même temps sans ralentir.

□ Concepts Intermédiaires

1. **Classes et POO**

```
class User:
    def __init__(self, email, password, nom):
        self.email = email
        self.password = generate_password_hash(password)
        self.nom = nom

    def verify_password(self, password):
        return check_password_hash(self.password, password)

    def is_admin(self):
        # Implémentation...
        return False

# Utilisation
user = User("test@cinema.com", "password123", "Dupont")
if user.verify_password("password123"):
    print("Bienvenue!")
```

2. **Exceptions (Try/Except)**

```
try:
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM users WHERE id = ?", (user_id,))

except sqlite3.Error as e:
    print(f"Erreur base de données: {e}")

except Exception as e:
    print(f"Erreur inattendue: {e}")

finally:
    conn.close()  # S'exécute toujours
```

3. **JSON**

Convertir entre Python (dict) et JSON (string).

```
import json

# Python dict → JSON string
user_data = {"name": "Alice", "age": 30}
json_string = json.dumps(user_data)  # '{"name": "Alice", "age": 30}'

# JSON string → Python dict
received = '{"title": "Dune", "price": 12.50}'
movie = json.loads(received)  # {'title': 'Dune', 'price': 12.50}

# Utilisé dans CinéMax pour les réservations
seat_categories_json = json.dumps(["adult", "child", "adult"])
```

4. **Lambda et Compréhensions**

```
# Lambda (fonction anonyme)
sorted_movies = sorted(MOVIES, key=lambda m: m["duration"])

# Compréhension de liste (très efficace)
adult_movies = [m for m in MOVIES if m["ratings"] == "12+"]
squares = [x**2 for x in range(10)]

# Compréhension de dictionnaire
prices = {movie: 12.50 for movie in ["Dune", "Moana 2"]}
```

5. **SQLite**

```
import sqlite3

conn = sqlite3.connect("cinema.db")
cursor = conn.cursor()

# Créer une table
cursor.execute('''
    CREATE TABLE users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        email TEXT UNIQUE NOT NULL,
        password TEXT NOT NULL
    )
''')

# Insérer
cursor.execute(
    "INSERT INTO users (email, password) VALUES (?, ?)",
    ("test@cinema.com", hashed_password)
)

# Récupérer
cursor.execute("SELECT * FROM users WHERE email = ?", ("test@cinema.com",))
user = cursor.fetchone()

# Mettre à jour
cursor.execute("UPDATE users SET email = ? WHERE id = ?", ("new@cinema.com", 1))

# Supprimer
cursor.execute("DELETE FROM users WHERE id = ?", (1,))

conn.commit()
conn.close()
```

6. **Hachage de mots de passe**

```
from werkzeug.security import generate_password_hash, check_password_hash

# Hachage (transforme le mot de passe en une chaîne aléatoire)
password = "test1234"
hashed = generate_password_hash(password)  # $2b$12$... (très sécurisé)

# Vérification
is_correct = check_password_hash(hashed, "test1234")  # True
is_correct = check_password_hash(hashed, "wrong")     # False
```

□ Concepts Fondamentaux

1. **Imports**

```
from flask import Flask, request, jsonify
from datetime import datetime, timedelta
import sqlite3
```

2. **Variables et Types**

```
# Dictionnaire
PRICES = {"adult": 12.50, "child": 8.00}

# Liste
movies = [{"title": "Dune"}, {"title": "Moana 2"}]

# String
email = "test@cinema.com"

# Booléen
is_admin = True
```

3. **Fonctions**

```
def get_available_dates(days=5):
    """Retourne les dates disponibles"""
    today = datetime.now()
    return [
        (today + timedelta(days=d)).strftime("%Y-%m-%d")
        for d in range(days)
    ]

# Appel
dates = get_available_dates(7)
```

4. **Listes et Dictionnaires**

```
# Listes
movies = ["Dune", "Moana 2"]
movies.append("Inside Out 2")

# Dictionnaires
user = {"email": "test@cinema.com", "id": 1}
email = user["email"]
user["age"] = 25  # Ajouter une clé
user.get("role", "user")  # Retourne "user" si pas de clé "role"
```