Compte Rendu Lab 1

Nom , Prenom : Khaled Essghaier

Class : DSI21

Pydantic :

Pydantic.py > ...

```python
from pydantic import BaseModel, EmailStr, field_validator
from pydantic_core.core_schema import FieldValidationInfo

class User(BaseModel):
    name: str
    email: EmailStr     # Validates that the email has a correct format
    account_id: int

    # Validator to ensure that account_id is strictly positive
    @field_validator("account_id")
    @classmethod
    def validate_account_id(cls, value: int, info: FieldValidationInfo) -> int:
        if value <= 0:
            raise ValueError(f"account_id must be positive: {value}")
        return value

# This line will raise a validation error due to the invalid email
try:
    user = User(name='Ali', email='ali', account_id=1234)
except Exception as e:
    print("Validation Error:", e)

# Creating a valid user
user = User(name='Ali', email='ali@gmail.com', account_id=1234)

# Display the Python model
print(user)

# Convert the model to JSON (string)
user_json_str = user.model_dump_json()
print(user_json_str)

# Convert to a Python dictionary
user_json_obj = user.model_dump()
print(user_json_obj)

# Example of creating an instance from a raw JSON string
import json

json_str = '{"name": "Ali", "email": "ali@gmail.com", "account_id": 1234}'
user = User.model_validate_json(json_str)
print(user)
```

```
  value is not a valid email address: An email address must have an @-sign. [type=value_error, input_value='ali', input_type=str]
name='Ali' email='ali@gmail.com' account_id=1234
{"name":"Ali","email":"ali@gmail.com","account_id":1234}
{'name': 'Ali', 'email': 'ali@gmail.com', 'account_id': 1234}
{'name': 'Ali', 'email': 'ali@gmail.com', 'account_id': 1234}
name='Ali' email='ali@gmail.com' account_id=1234
PS D:\DSI21\Semaistre 2\Python avancé\pydantic>
```

Description :

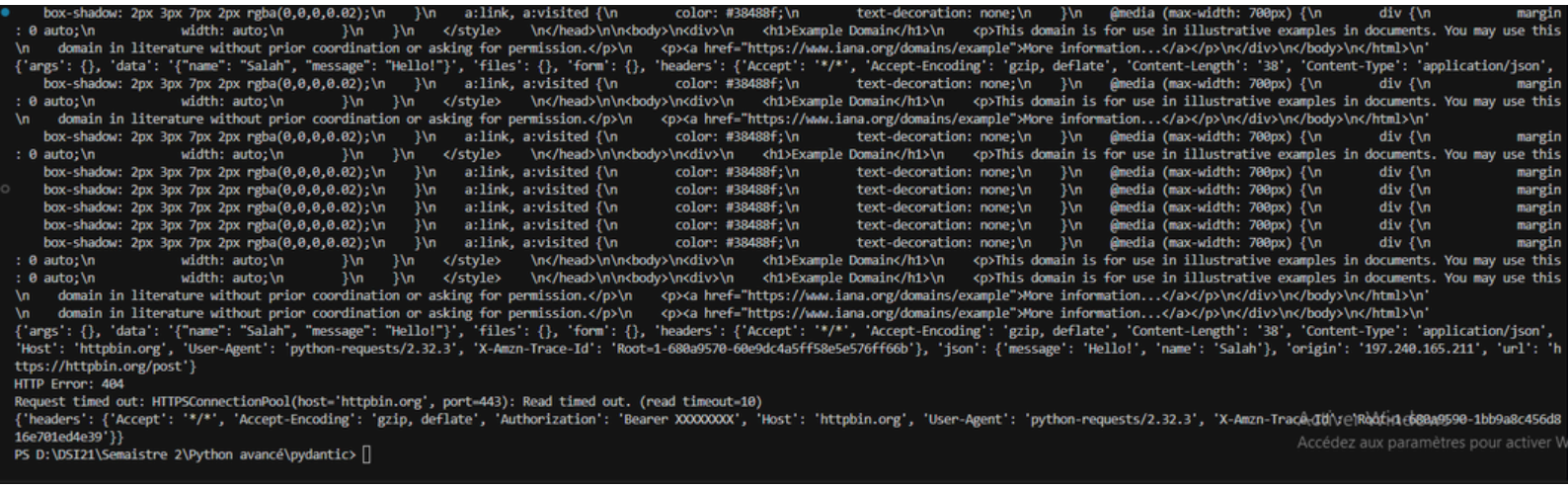
1. **User(BaseModel)** : Crée un modèle de données avec des types stricts.
2. **EmailStr** : Valide que l'email est bien formaté (ex: ali@gmail.com est OK, ali ne l'est pas).
3. **@field_validator** : Ajoute une règle personnalisée (ici, account_id doit être > 0).
4. **model_dump_json()** : Sérialise les données en chaîne JSON.
5. **model_dump()** : Donne les données sous forme de dictionnaire.
6. **model_validate_json()** : Parse une chaîne JSON et renvoie un objet User.

Ce code vérifie que les données d'un utilisateur sont correctes : l'email doit être valide et l'identifiant positif. Il permet aussi de convertir les données en JSON ou de créer un utilisateur à partir d'un JSON. C'est utile pour sécuriser les données dans une application.

Request :

Request.py > ...

```python
import requests
from bs4 import BeautifulSoup

# 1. GET request et affichage du contenu
response = requests.get("https://www.example.com")
print(response.content)  # Correction: 'repsonse' → 'response'

# 2. POST request avec des données JSON
data = {"name": "Salah", "message": "Hello!"}
url = "https://httpbin.org/post"
response = requests.post(url, json=data)
response_data = response.json()
print(response_data)  # Affiche les données envoyées

# 3. Vérification du statut HTTP
response = requests.get("https://httpbin.org/status/404")
if response.status_code != 200:
    print(f"HTTP Error: {response.status_code}")

# 4. Gestion du timeout
url = "https://httpbin.org/delay/10"
try:
    response = requests.get(url, timeout=10)
except requests.exceptions.Timeout as err:
    print("Request timed out:", err)

# 5. Requête avec jeton d'authentification
auth_token = "XXXXXXXX"
headers = {
    "Authorization": f"Bearer {auth_token}"
}
url = "https://httpbin.org/headers"
response = requests.get(url, headers=headers)
print(response.json())

# 6. Analyse du contenu HTML avec BeautifulSoup
url = "https://www.example.com"
response = requests.get(url)
soup = BeautifulSoup(response.content, "html.parser")

```



Description :

Ce script montre plusieurs exemples d'utilisation de la bibliothèque requests :

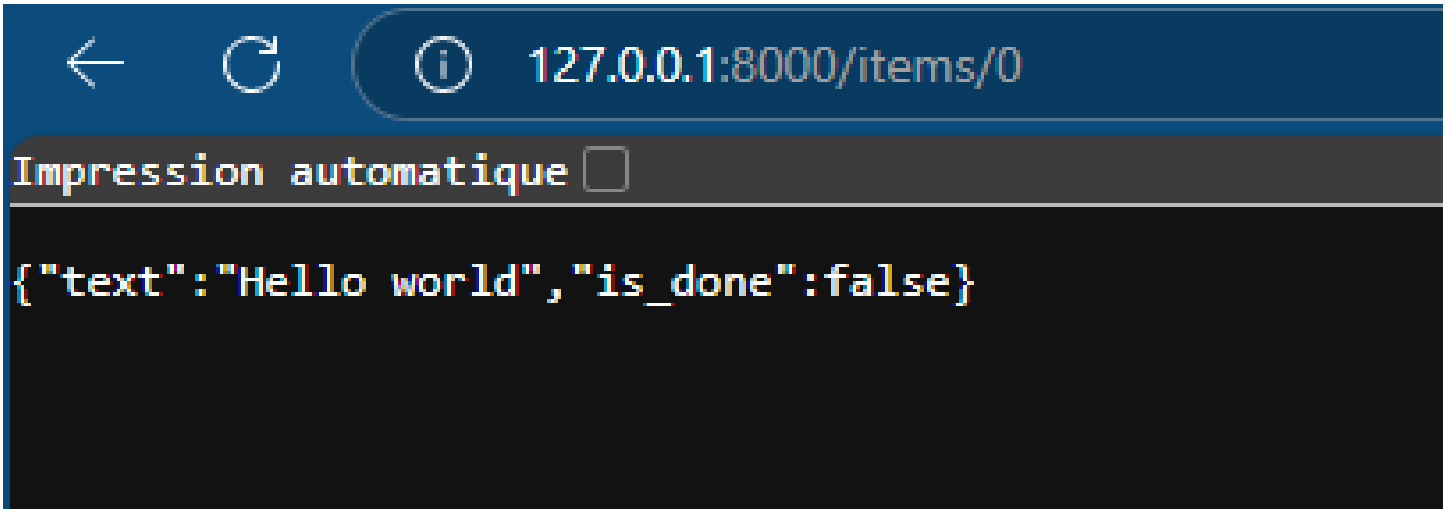
- Récupérer et afficher le contenu d'un site.
- Envoyer des données à un serveur avec POST.
- Vérifier les erreurs (comme une page 404).
- Gérer un délai d'attente (timeout).
- Ajouter un jeton d'authentification (Bearer token) dans l'en-tête HTTP.
- Et enfin, utiliser BeautifulSoup pour lire et analyser le HTML d'une page.

Ce code montre comment utiliser Python pour communiquer avec des sites web. Il permet de lire le contenu d'une page, envoyer des données, vérifier si une page existe, gérer les erreurs de connexion, et utiliser un jeton pour s'authentifier. Il utilise aussi BeautifulSoup pour lire le contenu HTML d'une page. C'est utile pour créer des scripts qui récupèrent ou envoient des informations automatiquement sur internet.

FastAPI :

main.py > ...

```python
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel

app = FastAPI()

# Modèle de données
class Item(BaseModel):
    text: str
    is_done: bool = False

# Liste pour stocker les items
items: list[Item] = []

# Route principale
@app.get("/")
def root():
    return {"Hello": "World"}

# Créer un nouvel item
@app.post("/items", response_model=Item)
def create_item(item: Item):
    items.append(item)
    return item

# Récupérer un item par son ID
@app.get("/items/{item_id}", response_model=Item)
def get_item(item_id: int):
    if 0 <= item_id < len(items):
        return items[item_id]
    raise HTTPException(status_code=404, detail=f"Item {item_id} not found")

# Lister plusieurs items
@app.get("/items", response_model=list[Item])
def list_items(limit: int = 10):
    return items[:limit]

```



Description :

Ce code crée une petite application web avec FastAPI. On peut :

- Accéder à la page d'accueil ("/").
- Ajouter un nouvel élément (appelé "item").
- Voir un élément en donnant son numéro.
- Lister plusieurs éléments (par exemple les 10 premiers).

Streamlit :

main.py > ...

```python
import streamlit as st
import pandas as pd
import numpy as np
import math  # Importer math pour math.ceil

# Titre de l'application
st.write('# Hello World 👋')

# Champ de texte pour saisir un film préféré
x = st.text_input('🎬 Quel est ton film préféré ?')

# Affiche le film saisi
if x:
    st.write(f"🍿 Ton film préféré est : **{x}**")

# Bouton interactif
if st.button("Clique ici !"):
    st.write("🚀 Tu as cliqué sur le bouton.")

# Titres et mise en forme
st.write("## Ceci est un titre H2")

st.markdown("*Streamlit* est **vraiment** ***génial***.")
st.markdown('''
:red[Streamlit] :orange[peut] :green[écrire] :blue[du texte] :violet[en]
:gray[beaux] :rainbow[couleurs] et :blue-background[mettre en surbrillance].
''')
st.markdown("Voici un bouquet : :tulip::cherry_blossom::rose::hibiscus::sunflower::blossom:")

# Exemple de texte multi-ligne
multi = '''Si tu termines une ligne avec deux espaces,
un retour à la ligne doux est utilisé.

Deux (ou plusieurs) sauts de ligne donnent un retour dur.
'''
st.markdown(multi)

# Lecture d'un fichier CSV (vérifie que le fichier "movies.csv" est dans le même dossier)
try:
    data = pd.read_csv("movies.csv")
    st.write("### 📄 Données du fichier `movies.csv`")
    st.dataframe(data)
except FileNotFoundError:
    st.error("❌ Fichier 'movies.csv' introuvable. Vérifie qu'il est bien dans le dossier.")
```

```python
# Données aléatoires pour les graphiques
chart_data = pd.DataFrame(
    np.random.randn(20, 3),
    columns=["a", "b", "c"]
)

st.write("### 📊 Diagramme en barres")
st.bar_chart(chart_data)

st.write("### 📈 Courbe linéaire")
st.line_chart(chart_data)

# Calculatrice de remboursement de prêt hypothécaire
st.title("Mortgage Repayments Calculator")

st.write("### Input Data")
col1, col2 = st.columns(2)  # Affichage en 2 colonnes
# Valeurs par défaut ou valeurs minimales/maximales
home_value = col1.number_input("Home Value", min_value=0, value=500000)
deposit = col1.number_input("Deposit", min_value=0, value=100000)
interest_rate = col2.number_input("Interest Rate (in %)", min_value=0.0, value=5.5)
loan_term = col2.number_input("Loan Term (in years)", min_value=1, value=30)

# Calcul des paiements
loan_amount = home_value - deposit
monthly_interest_rate = (interest_rate / 100) / 12
number_of_payments = loan_term * 12
monthly_payment = (
    loan_amount
    * (monthly_interest_rate * (1 + monthly_interest_rate) ** number_of_payments)
    / ((1 + monthly_interest_rate) ** number_of_payments - 1)
)

# Affichage des paiements
total_payments = monthly_payment * number_of_payments
total_interest = total_payments - loan_amount

st.write("### Repayments")
col1, col2, col3 = st.columns(3)  # Création de 3 colonnes
col1.metric(label="Monthly Repayments", value=f"${monthly_payment:,.2f}")
col2.metric(label="Total Repayments", value=f"${total_payments:,.0f}")
col3.metric(label="Total Interest", value=f"${total_interest:,.0f}")

```

```python
    # Création du plan de remboursement
    schedule = []
    remaining_balance = loan_amount

    for i in range(1, number_of_payments + 1):
        interest_payment = remaining_balance * monthly_interest_rate
        principal_payment = monthly_payment - interest_payment
        remaining_balance -= principal_payment
        year = math.ceil(i / 12)  # Calcul de l'année du prêt
        schedule.append(
            [
                i,
                monthly_payment,
                principal_payment,
                interest_payment,
                remaining_balance,
                year,
            ]
        )

    df = pd.DataFrame(
        schedule,
        columns=["Month", "Payment", "Principal", "Interest", "Remaining Balance", "Year"],
    )

    # Affichage du plan de paiement sous forme de graphique
    st.write("### Payment Schedule")
    payments_df = df[["Year", "Remaining Balance"]].groupby("Year").min()
    st.line_chart(payments_df)

```

Hello World 👋

🎬 Quel est ton film préféré ?

Clique ici !

Ceci est un titre H2

Streamlit est **vraiment *génial***.

Streamlit peut écrire du texte en beaux couleurs et `mettre en surbrillance`.

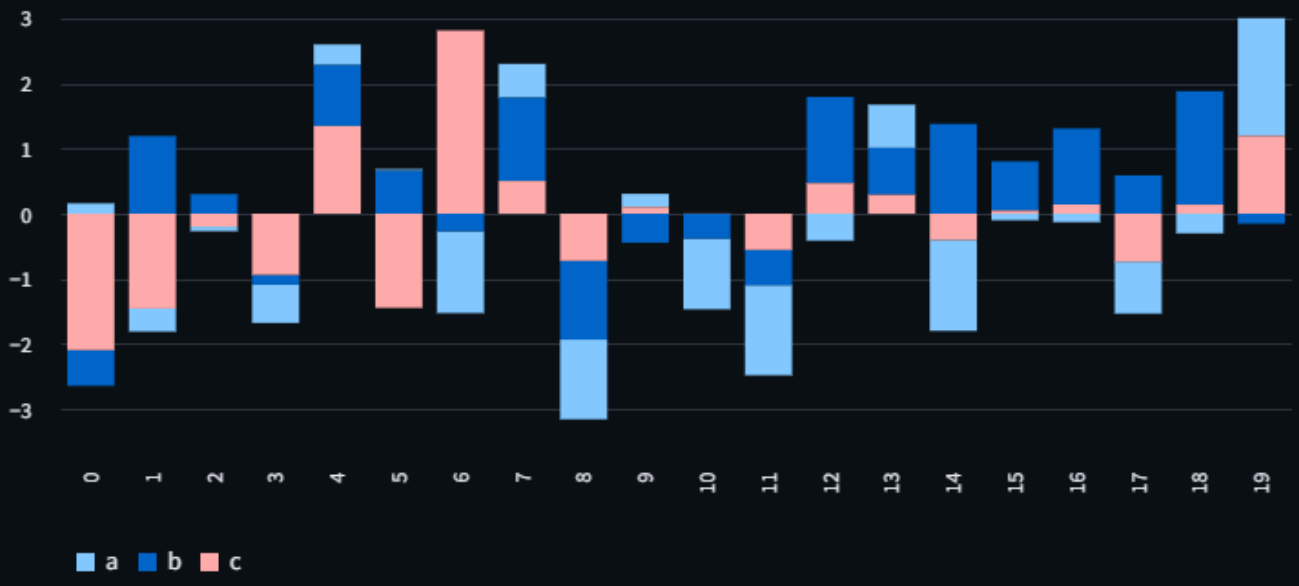
Voici un bouquet : 🌷 🌸 🌹 🌺 🌻 💮

Si tu termines une ligne avec deux espaces,
un retour à la ligne doux est utilisé.

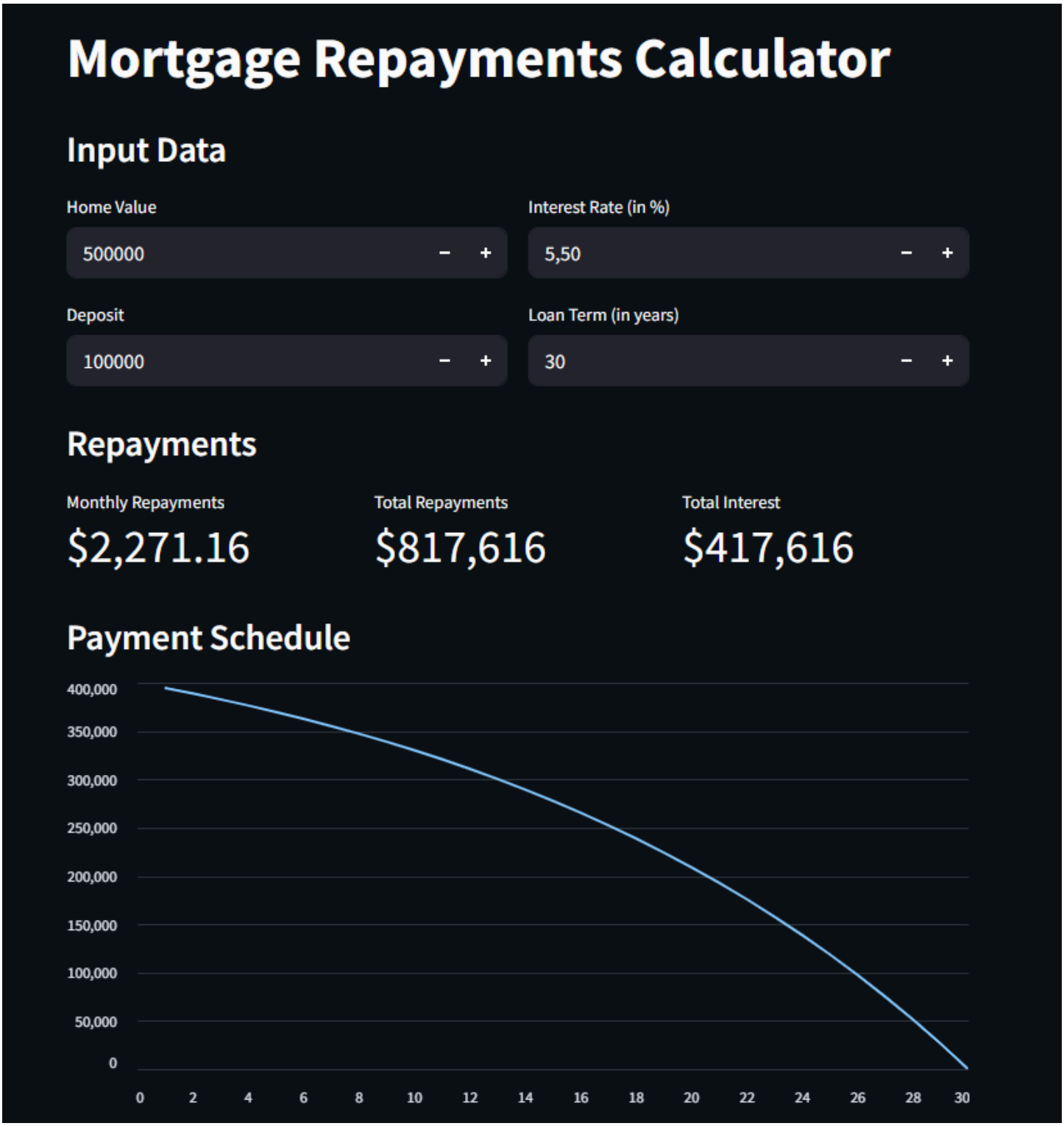
Deux (ou plusieurs) sauts de ligne donnent un retour dur.

❌ Fichier 'movies.csv' introuvable. Vérifie qu'il est bien dans le dossier.

Diagramme en barres



Courbe linéaire



Description :

Importation des bibliothèques nécessaires :

- **Streamlit** : Pour créer une interface web interactive.
- **Pandas et NumPy** : Pour manipuler des données et générer des valeurs aléatoires.

Affichage de contenu interactif :

- Utilisation de **st.write()** pour afficher du texte et des titres.

- Utilisation de **st.text_input()** pour capturer l'entrée de l'utilisateur (ici, le film préféré).
- Utilisation de **st.button()** pour créer un bouton interactif et afficher un message lorsque l'utilisateur clique dessus.

Mise en forme avec Markdown :

- Utilisation de **st.markdown()** pour afficher du texte avec une mise en forme avancée (italiques, gras, emojis, couleurs).

Lecture et affichage de données CSV :

- Chargement d'un fichier CSV avec **Pandas** et affichage des données dans un tableau avec **st.dataframe()**.

Génération de graphiques :

- Génération de graphiques avec des données aléatoires (barres et courbes) grâce à **st.bar_chart()** et **st.line_chart()**.

Calculatrice de remboursements hypothécaires :

- Saisie de données comme la valeur de la maison, le dépôt, le taux d'intérêt et la durée du prêt via **number_input()**.
- Calcul des paiements mensuels du prêt en utilisant la formule des prêts hypothécaires.
- Affichage des résultats sous forme de **st.metric()** pour indiquer les paiements mensuels, le total des paiements et des intérêts.

Plan de remboursement :

- Création d'un plan de remboursement mois par mois avec le principal, l'intérêt, et le solde restant.
- Affichage du plan sous forme de graphique pour visualiser l'évolution du solde restant sur la durée du prêt.

Le code crée une application Streamlit permettant à l'utilisateur d'interagir avec différents éléments comme des champs de texte, des boutons, des graphiques et une calculatrice de remboursement hypothécaire. Tout cela est présenté de manière visuellement agréable et interactive, offrant à l'utilisateur une expérience fluide et dynamique.