Elections américaines (Version Web) - v1.1

Ce projet a pour objectif de développer une application web Flask permettant d'explorer les résultats d'une élection américaine par état, avec des données fournies au format JSON. Vous manipulerez les routes statiques et dynamiques pour afficher des informations variées sur les résultats des élections, avec des fonctions adaptées pour récupérer et filtrer les données.

Structure des données JSON

Les données concernant les états américains sont fournies sous forme de liste d'objets JSON. Voici la structure de chaque objet :

```json
{
    "article": "l'",
    "court": "Arizona",
    "nom": "Arizona",
    "typo_2020": "d+",
    "grd_2020": "11",
    "2020": "d",
    "typo": "-",
    "chgt_grd": null,
    "grd": "11",
    "vainqueur": "r",
    "champlibre": null,
    "code": "AZ"
}
```

Explication des champs :

- `article` : Article grammatical pour le nom de l'état (ex. "l'", "le").
- `court` : Nom court de l'état.
- `nom` : Nom complet de l'état.
- `typo_2020` : Tendance politique en 2020 (par exemple, "d+" pour démocrate).
- `grd_2020` : Nombre de grands électeurs en 2020.
- `2020` : Parti vainqueur en 2020 ("d" pour démocrate, "r" pour républicain).
- `typo` : Tendance politique actuelle.
- `chgt_grd` : Variation dans le nombre de grands électeurs par rapport à 2020 (null si inchangé).
- `grd` : Nombre actuel de grands électeurs.
- `vainqueur` : Parti vainqueur de l'élection actuelle.
- `champlibre` : Champ libre pour des données supplémentaires (actuellement null).
- `code` : Code postal de l'état (par exemple, "AZ" pour l'Arizona).

Fonctionnalités à implémenter

1. Route : `/vainqueur`

Créez une route statique `/vainqueur` qui affiche le vainqueur de l'élection actuelle (le parti ayant remporté le plus d'états). Pour cela :

- Parcourez les données JSON pour compter le nombre d'états remportés par chaque parti (`r`, `d`, ou autres).
- Affichez un message indiquant le vainqueur (par exemple : "Le vainqueur est le parti républicain avec 27 états.").

2. Route dynamique : `/etats/<code_etat>`

Créez une route dynamique `/etats/<code_etat>` qui affiche les détails d'un état spécifique, en utilisant son code (par exemple, "AZ" pour l'Arizona). Pour cela :

1. Créez une **fonction** qui recherche l'état correspondant dans les données JSON.
2. Créez un **template HTML** qui affiche toutes les données de l'état sous forme de tableau (ou autre format structuré).
3. Gérez les cas où le code de l'état n'est pas trouvé en affichant un message d'erreur (ex. : "L'état demandé n'existe pas.").

3. Route dynamique : `/partis/<code_etat>`

Créez une route dynamique `/partis/<code_etat>` qui affiche tous les états ayant voté pour un parti spécifique. Pour cela :

1. Filtrez les états dans les données JSON où le champ `vainqueur` correspond au code du parti (`r`, `d`, etc.).
2. Créez un **template HTML** qui affiche ces états sous forme de liste ou tableau.
3. Si aucun état n'a voté pour ce parti, affichez un message d'information (ex. : "Aucun état n'a voté pour ce parti.").

4. Route : `/bascules`

Créez une route `/bascules` qui affiche la liste des états ayant basculé d'un parti à un autre depuis 2020. Un état est considéré comme ayant "basculé" si le champ `2020` (vainqueur en 2020) est différent du champ `vainqueur` (vainqueur actuel). Pour cela :

1. Filtrez les états ayant basculé.
2. Réutilisez le template HTML de la route `/etats/<code_etat>` pour afficher les informations des états.

5. Route : `/electeurs/<int:nombre>`

Créez une route dynamique `/electeurs/<int:nombre>` qui affiche les états ayant au moins un certain nombre de grands électeurs. Pour cela :

1. Filtrez les états où le champ `grd` est supérieur ou égal à `nombre`.
2. Créez un **template HTML** pour afficher ces états.
3. Si aucun état ne correspond, affichez un message d'information (ex. : "Aucun état n'a au moins 15 grands électeurs.").

Fonctionalié `BONUS`: affichez la carte des états

Exercice en cours d'élaboration. Vous pouvez essayer de le résoudre avec l'IA, même si l'énoncé n'est pas encore complet.

Vous allez maintenant afficher une carte des états colorée en fonction des résultats des élections:

- sur la page qui affiche les résultats globaux des élections (/)
- sur la page qui affiche les résultats d'un état (/etat/<code_etat>)

Pour ce faire, téléchargez les fichiers `us2.json` et `us2.svg` en pièce jointe.

Le but de l'exercice est de créer une version dynamique du fichier SVG en utilisant les données du fichier JSON.

- Créez un fichier `map.svg` dans le dossier `templates`
- Insérez-le dans votre template jinja grâce à l'instruction `{%include "map.svg" %}`.
- Les routes qui affichent la map doivent charger le fichier `us2.json`.

Points importants

1. Organisation du projet

Organisez votre projet comme suit :

```
elections/
│
├── static/
│   └── css/    (styles pour vos templates, facultatif)
│
├── templates/
│   ├── index.html  (page d'accueil ou autre)
│   ├── vainqueur.html
│   ├── etat.html
│   └── liste_etats.html
│
├── app.py
└── data.json  (fichier contenant les données JSON)
```

2. Chargement des données

Chargez les données JSON dans Flask au démarrage de l'application en les lisant depuis un fichier externe (`data.json`) :

```python
import json

with open('data.json', 'r') as file:
    data = json.load(file)
```

Vous pourrez ensuite parcourir ces données avec des boucles ou des filtres dans vos routes.

Filtrer des listes en Python

Filtrer une liste en Python signifie créer une nouvelle liste contenant uniquement les éléments qui répondent à une condition donnée. Voici comment le faire de deux façons :

1. "**À la main**", c'est-à-dire avec une boucle et des constructions basiques.
2. **De la "bonne façon"**, en utilisant les outils modernes et concis de Python.

1. Filtrer une liste "à la main"

Cette méthode consiste à utiliser une boucle `for` pour parcourir chaque élément de la liste, vérifier si l'élément satisfait la condition, et ajouter cet élément à une nouvelle liste si la condition est remplie.

Exemple : Filtrer les nombres pairs d'une liste

```python
# Liste initiale
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Filtrer "à la main"
filtered_numbers = []
for number in numbers:
    if number % 2 == 0:  # Vérifie si le nombre est pair
        filtered_numbers.append(number)

print(filtered_numbers)  # Résultat : [2, 4, 6, 8, 10]
```

Points à noter :

- Cette méthode est facile à comprendre, surtout pour les débutants.
- Cependant, elle est **plus longue et moins idiomatique** pour des utilisateurs expérimentés de Python.
- Elle nécessite explicitement d'initialiser une liste vide et d'ajouter des éléments un par un.

2. Filtrer une liste de la "bonne façon"

Python propose des outils modernes et élégants pour filtrer une liste, comme les **compréhensions de liste** ou la fonction `filter()`. Ces méthodes sont concises, lisibles et plus rapides à écrire.

Méthode 1 : Compréhension de liste

Une **compréhension de liste** permet de combiner une boucle et une condition en une seule ligne.

```
# Liste initiale
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Filtrage avec une compréhension de liste
filtered_numbers = [number for number in numbers if number % 2 == 0]

print(filtered_numbers)  # Résultat : [2, 4, 6, 8, 10]
```

- **Avantages** :
  - Plus concise et lisible pour ceux qui maîtrisent Python.
  - Performante, car optimisée dans l'interpréteur Python.
  - Idiomatique (conforme aux bonnes pratiques de Python).

Méthode 2 : Utiliser `filter()`

La fonction `filter()` est une fonction intégrée qui prend deux arguments :

1. Une fonction qui retourne `True` ou `False` (la condition).
2. La liste à filtrer.

Elle retourne un **itérable** contenant uniquement les éléments qui répondent à la condition.

```
# Liste initiale
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Fonction pour tester si un nombre est pair
def is_even(number):
    return number % 2 == 0

# Filtrage avec filter()
filtered_numbers = list(filter(is_even, numbers))

print(filtered_numbers)  # Résultat : [2, 4, 6, 8, 10]
```

Vous pouvez également utiliser une **lambda** pour rendre le tout plus concis :

```
filtered_numbers = list(filter(lambda number: number % 2 == 0, numbers))
print(filtered_numbers)  # Résultat : [2, 4, 6, 8, 10]
```

- **Avantages** :
  - Lisible et bien adapté aux situations nécessitant des fonctions déjà définies.
  - Utile pour intégrer directement dans des chaînes de traitement fonctionnel (avec `map`, `reduce`, etc.).

Résumé : Quelle méthode utiliser ?

Méthode	Cas d'utilisation
Boucle "à la main"	Pour les débutants ou pour des cas très spécifiques nécessitant une logique complexe dans la boucle.

Méthode	Cas d'utilisation
Compréhension de liste	**À privilégier** : concis, lisible, idiomatique pour la plupart des cas.
`filter()`	À utiliser si vous avez déjà une fonction séparée pour la condition, ou dans des traitements en chaîne.

En général, la **compréhension de liste** est la meilleure option : elle est rapide à écrire, facile à comprendre, et s'intègre bien dans le style moderne de Python.

Suggestions pour aller plus loin

1. **Ajoutez des styles CSS** pour améliorer l'apparence des pages.
2. **Implémentez un système de recherche** permettant aux utilisateurs de rechercher un état par nom.
3. **Créez des graphiques** pour visualiser les résultats de l'élection (ex. : nombre d'états gagnés par chaque parti).

Objectifs pédagogiques

- Maîtriser les routes statiques et dynamiques dans Flask.
- Manipuler des données JSON et les afficher dans des templates HTML.
- Utiliser des filtres et des boucles pour sélectionner et présenter des données pertinentes.
- Structurer un projet Flask avec des templates réutilisables.

Ce projet vous permettra de consolider vos compétences en Flask tout en apprenant à manipuler et afficher des données de manière efficace et claire.