Projet : Un site web avec une base de données

Ce document est une adaptation du site pixees : https://pixees.fr/informatiquelycee/n_site/nsi_prem.html

1 Manipuler des SGBD SQL avec Python

1.1 Le module Python sqlite3

A faire vous même 1.

Après avoir créé un répertoire `projet_bd`. Créez, à l'aide de spyder, un fichier Python (à vous de choisir le nom) puis saisissez et exécutez le programme suivant :

```
import sqlite3

conn = sqlite3.connect('baseDonnees.db')
cur = conn.cursor()

cur.execute("CREATE TABLE LIVRES(id INT, titre TEXT, auteur TXT,
ann_publi INT, note INT)")

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

Analysons le programme ci-dessus :
Ce programme va vous permettre de vous "connecter" à une base de données (si cette dernière n'existe pas, elle sera créée). Ensuite nous créons une table (une relation) nommée LIVRES.
Analysons le programme ligne par ligne :

```
import sqlite3
```

Nous commençons par importer la bibliothèque sqlite3. Cette bibliothèque va nous permettre d'effectuer des requêtes SQL sur une base de données. Nous utiliserons le SGBD SQLite.

```
conn = sqlite3.connect('baseDonnees.db')
cur = conn.cursor()
```

Nous créons un objet de type "connection" (conn) qui va nous permettre d'interagir avec la base de données "baseDonnees.db". Vous devriez donc avoir un fichier "baseDonnees.db" dans le même répertoire que votre fichier Python.

Nous créons ensuite un objet de type "cursor" à partir de l'objet de type "connection". Cet objet de type "cursor" va nous permettre de manipuler la base de données et d'obtenir des résultats lorsque nous effectuerons des requêtes.

```
cur.execute("CREATE TABLE LIVRES(id INT, titre TEXT, auteur TXT,
ann_publi INT, note INT)")
```

La méthode "execute" de notre objet de type "cursor" nous permet d'effectuer une requête SQL. Cette requête SQL est en tout point identique à ce que nous avons vu dans le cours sur les bases de données.

```
conn.commit()
```

Pour véritablement exécuter les requêtes, il est nécessaire d'appliquer la méthode "commit" à l'objet de type "connection".

```
cur.close()
conn.close()
```

Avant de terminer le programme, il nécessaire de "fermer" l'objet de type "cursor" et l'objet de type "connection".
Nous allons systématiquement retrouver cette structure dans nos futurs programmes :

- création d'un objet de type "connection"
- création d'un objet de type "cursor"
- préparation d'une ou plusieurs requête(s) (méthode "execute" sur l'objet de type "cursor")

- exécution réelle des requêtes (méthode "commit" sur l'objet de type "connection")
- "fermeture" de l'objet de type "cursor" puis de l'objet de type "connection"

Si vous exécutez une deuxième fois le programme proposé au "À faire vous-même 1", vous aurez droit à une erreur : "OperationalError: table LIVRES already exists". Afin d'éviter ce genre de problème, il est possible de modifier le programme afin que la requête de création de la table LIVRES ne se fasse pas si la table LIVRES existe déjà :

A faire vous même 2.

Après avoir effacé le fichier `baseDonnees.db`, saisissez le programme ci-dessous, puis exécutez-le

```
import sqlite3

conn = sqlite3.connect('baseDonnees.db')
cur = conn.cursor()

cur.execute("CREATE TABLE IF NOT EXISTS LIVRES(id INT, titre TEXT,
auteur TXT, ann_publi INT, note INT)")

conn.commit()

cur.close()
conn.close()
```

A faire vous même 3.

Saisissez le programme ci-dessous, puis exécutez-le

```
import sqlite3

conn = sqlite3.connect('baseDonnees.db')
cur = conn.cursor()

cur.execute("CREATE TABLE IF NOT EXISTS LIVRES(id INT, titre TEXT,
auteur TXT, ann_publi INT, note INT)")
cur.execute("INSERT INTO LIVRES(id,titre,auteur,ann_publi,note)
VALUES(1,'1984','Orwell',1949,10)")

conn.commit()

cur.close()
conn.close()
```

Rien de particulier à signaler, la requête INSERT est identique à ce qui a été vu dans le cours sur les bases de données.

A faire vous même 4.

Après avoir effacé le fichier "baseDonnees.db", saisissez le programme ci-dessous, puis exécutez-le

```
import sqlite3

conn = sqlite3.connect('baseDonnees.db')
cur = conn.cursor()

data = (1,'1984','Orwell',1949,10)

cur.execute("CREATE TABLE IF NOT EXISTS LIVRES(id INT, titre TEXT,
auteur TXT, ann_publi INT, note INT)")
cur.execute("INSERT INTO LIVRES(id,titre,auteur,ann_publi,note)
VALUES(?, ?, ?, ?, ?)", data)
conn.commit()

cur.close()
conn.close()
```

Nous avons créé un tuple (data) contenant toutes les informations. En effet, la méthode "execute" peut prendre un deuxième paramètre un tuple contenant les données à insérer. Les points d'interrogation présents dans la requête indiquent l'emplacement des données à insérer. Le

premier ? sera remplacé par le premier élément du tuple (dans notre cas 1), le deuxième ? sera remplacé par le deuxième élément du tuple (dans notre cas '1984') et ainsi de suite...

A faire vous même 5.

Si l'on désire insérer plusieurs données, il est possible de regrouper toutes les données à insérer dans un tableau et d'utiliser la méthode "executemany" à la place de la méthode "execute".
Après avoir effacé le fichier "baseDonnees.db", saisissez le programme ci-dessous, puis exécutez-le

```
import sqlite3

conn = sqlite3.connect('baseDonnees.db')
cur = conn.cursor()

datas = [
        (1,'1984','Orwell',1949,10),
        (2,'Dune','Herbert',1965,8),
        (3,'Fondation','Asimov',1951,9),
        (4,'Le meilleur des mondes','Huxley',1931,7),
        (5,'Fahrenheit 451','Bradbury',1953,7),
        (6,'Ubik','K.Dick',1969,9),
        (7,'Chroniques martiennes','Bradbury',1950,8),
        (8,'La nuit des temps','Barjavel',1968,7),
        (9,'Blade Runner','K.Dick',1968,8),
        (10,'Les Robots','Asimov',1950,9),
        (11,'La Planète des singes','Boulle',1963,8),
        (12,'Ravage','Barjavel',1943,8),
        (13,'Le Maître du Haut Château','K.Dick',1962,8),
        (14,'Le monde des Ã','Van Vogt',1945,7),
        (15,'La Fin de l'éternité','Asimov',1955,8),
        (16,'De la Terre à la Lune','Verne',1865,10)
 ]

cur.execute("CREATE TABLE IF NOT EXISTS LIVRES(id INT, titre TEXT,
auteur TXT, ann_publi INT, note INT)")
cur.executemany("INSERT INTO LIVRES(id,titre,auteur,ann_publi,note)
VALUES(?, ?, ?, ?, ?)", datas)
conn.commit()

cur.close()
conn.close()
```

A faire vous même 6.

Il n'est pas très pratique d'avoir à gérer l'id (clé primaire). En effet, si je veux ajouter un nouveau livre, il faudra que je connaisse l'id du précédent (incrémentation de l'id). Heureusement, il est possible d'automatiser cette incrémentation.
Après avoir effacé le fichier "baseDonnees.db", saisissez le programme ci-dessous, puis exécutez-le

```
import sqlite3

conn = sqlite3.connect('baseDonnees.db')
cur = conn.cursor()

datas = [
        ('1984','Orwell',1949,10),
        ('Dune','Herbert',1965,8),
        ('Fondation','Asimov',1951,9),
        ('Le meilleur des mondes','Huxley',1931,7),
        ('Fahrenheit 451','Bradbury',1953,7),
        ('Ubik','K.Dick',1969,9),
        ('Chroniques martiennes','Bradbury',1950,8),
        ('La nuit des temps','Barjavel',1968,7),
        ('Blade Runner','K.Dick',1968,8),
        ('Les Robots','Asimov',1950,9),
        ('La Planète des singes','Boulle',1963,8),
```

```
            ('Ravage','Barjavel',1943,8),
            ('Le Maître du Haut Château','K.Dick',1962,8),
            ('Le monde des Ā','Van Vogt',1945,7),
            ('La Fin de l'éternité','Asimov',1955,8),
            ('De la Terre à la Lune','Verne',1865,10)
    ]

    cur.execute("CREATE TABLE IF NOT EXISTS LIVRES(id INTEGER PRIMARY
KEY AUTOINCREMENT UNIQUE, titre TEXT, auteur TXT, ann_publi INT, note
INT)")
    cur.executemany("INSERT INTO LIVRES(titre,auteur,ann_publi,note)
VALUES(?, ?, ?, ?)", datas)
    conn.commit()

    cur.close()
    conn.close()
```

Ouvrez le fichier "baseDonnees.db" à l'aide du logiciel "DB Browser for SQLite" et vérifiez que les données ont bien été ajoutées à la table LIVRES.
Vous pouvez constater que nous avons bien l'attribut "id", même si ce dernier n'a pas été renseigné dans les données (absence d'id dans le tableau datas). Désormais l'id sera incrémenté automatiquement grâce au "id INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE" (attention il est nécessaire d'utiliser INTEGER à la place du INT habituel) présent dans la requête de création de la table LIVRES. Attention, de bien penser à supprimer un ? dans la requête d'insertion (chaque tuple contient maintenant 4 éléments (nous en avions 5 quand l'id n'était pas géré automatiquement)).

A faire vous même 7.

Il est tout à fait possible de rajouter une nouvelle donnée :
Saisissez le programme ci-dessous, puis exécutez-le

```
    import sqlite3

    conn = sqlite3.connect('baseDonnees.db')
    cur = conn.cursor()
    nvx_data = ('Hypérion','Simmons',1989,8)

    cur.execute("INSERT INTO LIVRES(titre,auteur,ann_publi,note)
VALUES(?, ?, ?, ?)", nvx_data)
    conn.commit()

    cur.close()
    conn.close()
```

A faire vous même 8.

Il est possible de modifier des données déjà présentes dans la table.

```
    import sqlite3

    conn = sqlite3.connect('baseDonnees.db')
    cur = conn.cursor()

    modif = (7, 'Hypérion')
    cur.execute('UPDATE LIVRES SET note = ? WHERE titre = ?', modif)
    conn.commit()

    cur.close()
    conn.close()
```

A faire vous même 9.

Il est aussi possible de supprimer une donnée :

```
    import sqlite3

    conn = sqlite3.connect('baseDonnees.db')
    cur = conn.cursor()

    suppr = ('Hypérion',)
```

```
cur.execute('DELETE FROM LIVRES WHERE titre = ?', suppr)
conn.commit()

cur.close()
conn.close()
```

A faire vous même 10.

Enfin, pour terminer cette introduction sur l'utilisation de sqlite en Python, nous devons nous intéresser aux requêtes de type "SELECT" :

```
import sqlite3

conn = sqlite3.connect('baseDonnees.db')
cur = conn.cursor()

cur.execute('SELECT * FROM LIVRES')
conn.commit()

liste = cur.fetchall()

cur.close()
conn.close()
```

À l'aide de la console, déterminez la valeur référencée par la variable liste

A faire vous même 11.

Il est possible d'utiliser les points d'interrogation dans une requête de type SELECT :
Saisissez le programme ci-dessous, puis exécutez-le

```
import sqlite3

conn = sqlite3.connect('baseDonnees.db')
cur = conn.cursor()

recherche = (1960, 8)

cur.execute('SELECT titre FROM LIVRES WHERE ann_publi < ? AND note > ?', recherche)
conn.commit()

liste = cur.fetchall()

cur.close()
conn.close()
```

À l'aide de la console, déterminez la valeur référencée par la variable liste

2 Propulsez un site web avec python

Un serveur web (aussi appelé serveur HTTP) permet de répondre à une requête HTTP effectuée par un client (très souvent un navigateur web).
Nous allons travailler avec le serveur web qui est installé sur votre ordinateur. Nous allons donc avoir une configuration un peu particulière puisque le client et le serveur vont se trouver sur la même machine. Cette configuration est classique lorsque l'on désire effectuer de simples tests.
Nous aurons donc 2 logiciels sur le même ordinateur : le client (navigateur web) et le serveur (serveur web), ces 2 logiciels vont communiquer en utilisant le protocole HTTP. Nous allons travailler avec le framework Python Flask. Ce framework va nous permettre de générer des pages web côté serveur, il possède son propre serveur web.
À noter qu'il est aussi possible d'utiliser d'autres langages côté serveur : Java, C# et surtout PHP.
Nous allons commencer par un cas très simple où le serveur va renvoyer au client une simple page HTML statique.

A faire vous même 1

1. Dans votre répertoire personnel, créez un répertoire nommé `flask`.
2. À l'aide de Spyder, créez un fichier Python `views.py` (ce fichier devra être sauvegardé dans le répertoire `flask` précédemment créé).
3. Saisissez le code suivant dans le fichier `views.py`

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def index():
  return "<p>Tout fonctionne parfaitement</p>"
app.run(debug=True)
```

   Attention ! Le module flask doit d'abord être importé.
4. Après avoir exécuté le programme ci-dessus, ouvrez votre navigateur web et tapez dans la barre d'adresse `localhost:5000`.
   Vous devriez voir la phrase "Tout fonctionne parfaitement" s'afficher dans votre navigateur.

Une petite explication s'impose à propos du `localhost:5000` : comme nous l'avons déjà dit, notre serveur et notre client se trouvent sur la même machine, avec le `localhost`, on indique au navigateur que le serveur web se trouve sur le même ordinateur que lui (on parle de machine locale).
Le `5000` indique le port.
Stoppez l'exécution du programme dans Spyder.
Essayons de comprendre en détail ce qui s'est passé :
En exécutant le programme Python ci-dessus, le framework Flask a lancé un serveur web. Ce serveur web attend des requêtes HTTP sur le port 5000. En ouvrant un navigateur web et en tapant `localhost:5000`, nous faisons une requête HTTP, le serveur web fourni avec Flask répond à cette requête HTTP en envoyant une page web contenant uniquement `<p>Tout fonctionne parfaitement</p>`.

Reprenons le programme Python ligne par ligne :

```
from flask import Flask
```

Nous importons la bibliothèque Flask

```
app = Flask(__name__)
```

Nous créons un objet app : cette ligne est systématique nécessaire.

```
@app.route('/')
```

Nous utilisons ici un décorateur (cette notion ne sera pas traitée en NSI). Vous devez juste comprendre la fonction qui suit ce décorateur ("index"), sera exécutée dans le cas où le serveur web recevra une requête HTTP avec une URL correspondant à la racine du site ('/'), c'est à dire, dans notre exemple, le cas où on saisie dans la barre d'adresse "localhost:5000/" (ou simplement "localhost:5000").

```
def index():
    return "<p>Tout fonctionne parfaitement</p>"
```

En cas de requête HTTP d'un client avec l'URL "/", le serveur renvoie vers le client une page HTML contenant uniquement la ligne " Tout fonctionne parfaitement ".

```
app.run(debug=True)
```

Cette ligne permet de lancer le serveur, elle sera systématiquement présente.

A faire vous même 2

Modifiez le fichier Python `views.py` :

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def index():
  return "<p>Tout fonctionne parfaitement</p>"
@app.route('/about')
def about():
  return "<p>Une autre page</p>"
app.run(debug=True)
```

Après avoir exécuté le programme ci-dessus, saisissez `localhost:5000/about` dans la barre d'adresse de votre navigateur.
Comme vous pouvez le constater, le serveur nous renvoie dans ce cas une autre page.

Écrire le code HTML qui devra être renvoyé au client dans le programme Python n'est pas très pratique, Flask propose une autre solution bien plus satisfaisante :

A faire vous même 3

Modifiez le fichier
Dans votre répertoire `flask`, créez un répertoire `templates`. Dans ce répertoire `templates`, créez un fichier `index.html`. Saisissez le code HTML ci-dessous dans ce fichier `index.html`

```
<!doctype html>
<html lang="fr">
        <head>
                <meta charset="utf-8">
                <title>Ma page</title>
        </head>
        <body>
          <h1>Mon super site</h1>
          <p>Tout fonctionne parfaitement</p>
        </body>
</html>
```

A faire vous même 4

Modifiez le programme `views.py` comme suit :

```
from flask import Flask, render_template
app = Flask(__name__)
@app.route('/')
def index():
    return render_template("index.html")
app.run(debug=True)
```

Relancez le programme Python et tapez `localhost:5000` dans la barre d'adresse de votre navigateur

Le serveur renvoie maintenant au client la page HTML correspondant au fichier `index.html` qui a été créé dans le répertoire `templates`. Attention, les fichiers HTML devront systématiquement se trouver dans un répertoire nommé `templates`.
N. B. le `debug=True` de la dernière ligne permet de modifier les fichiers HTML sans être obligé de redémarrer le programme `views.py`.
Pour l'instant notre site est statique : la page reste identique, quelles que soient les actions des visiteurs. Flask permet de créer des pages dynamiques :

- le client (le navigateur web) envoie une requête HTTP vers un serveur web
- en fonction de la requête reçue et de différents paramètres, Flask "fabrique" une page HTML différente
- le serveur web associé à Flask envoie la page nouvellement créée au client
- une fois reçue, la page HTML est affichée dans le navigateur web

A faire vous même 5

Modifiez le fichier `views.py` comme suit :

```
from flask import Flask, render_template
import datetime

app = Flask(__name__)

@app.route('/')
def index():
  date = datetime.datetime.now()
  h = date.hour
  m = date.minute
  s = date.second
  return render_template("index.html", heure = h, minute = m,
seconde = s)

app.run(debug=True)
```

Dans le programme ci-dessous nous importons le module `datetime` afin de pouvoir déterminer la date et l'heure courante.

```
date = datetime.datetime.now()
```

nous permet de récupérer la date et l'heure courante

```
h = date.hour
m = date.minute
s = date.second
```

Après l'exécution des 3 lignes ci-dessus, les variables h, m et s contiennent l'heure courante.
La fonction `render_template`

```
return render_template("index.html", heure = h, minute = m, seconde = s)
```

contient 3 paramètres, nous allons retrouver ces 3 paramètres dans le fichier HTML.

A faire vous même 6

Modifiez le fichier `index.html` comme suit :

```
<!doctype html>
<html lang="fr">
        <head>
                <meta charset="utf-8">
                <title>Utilisation de Flask</title>
        </head>
        <body>
          <h1>Mon super site</h1>
          <p>Le serveur fonctionne parfaitement, il est {{heure}} h
{{minute}} minutes et {{seconde}} secondes</p>
        </body>
</html>
```

Testez ces modifications en saisissant `localhost:5000` dans la barre de votre navigateur web.

Nous avons bien une page dynamique, puisqu'à chaque fois que vous actualisez la page dans votre navigateur, l'heure courante s'affiche : à chaque fois que vous actualisez la page, vous effectuez une nouvelle requête et en réponse à cette requête, le serveur envoie une nouvelle page HTML.

Nous allons maintenant nous intéresser à la gestion des formulaires.

A faire vous même 7

Modifiez le fichier `index.html` comme suit :

```
<!doctype html>
<html lang="fr">
    <head>
        <meta charset="utf-8">
        <title>Le formulaire</title>
    </head>
    <body>
      <form action="http://localhost:5000/resultat" method="post">
         <label>Nom</label> : <input type="text" name="nom" />
         <label>Prénom</label> : <input type="text" name="prenom" />
         <input type="submit" value="Envoyer" />
      </form>
    </body>
</html>
```

et créez un fichier `resultat.html` (dans le répertoire `templates`), ce fichier devra contenir le code suivant :

```
<!doctype html>
<html lang="fr">
   <head>
     <meta charset="utf-8">
     <title>Résultat</title>
   </head>
   <body>
      <p>Bonjour {{prenom}} {{nom}}, j'espère que vous allez bien.</p>
   </body>
</html>
```

A faire vous même 8

Modifiez le fichier `views.py` comme suit :

```
from flask import Flask, render_template, request
app = Flask(__name__)
@app.route('/')
def index():
  return render_template('index.html')
@app.route('/resultat',methods = ['POST'])
def resultat():
  result = request.form
  n = result['nom']
  p = result['prenom']
  return render_template("resultat.html", nom=n, prenom=p)
app.run(debug=True)
```

Après avoir relancé `views.py`, testez cet exemple en saisissant `localhost:5000`

Si vous saisissez, par exemple, `Martin` et `Sophie` dans les champs `nom` et `prenom` du formulaire, vous devriez obtenir le résultat suivant après avoir appuyé sur le bouton "Envoyer" :

Bonjour Sophie Martin, j'espère que vous allez bien.

Reprenons un par un les événements qui nous ont amenés à ce résultat :
Nous effectuons une requête HTTP avec l'URL "/", le serveur génère une page web à partir du fichier `index.html`, cette page, qui contient un formulaire (balise `<form action="http://localhost:5000/resultat" method="post">`) est envoyée vers le client. On remarque 2 attributs dans cette balise `form` : `action="http://localhost:5000/resultat"` et `method="post"`. Ces 2 attributs indiquent que le client devra effectuer une requête de type POST (la méthode POST a déjà été vue dans la partie consacrée au protocole HTTP) dès que l'utilisateur appuiera sur le bouton "Envoyer". Cette requête POST sera envoyée à l'URL `http://localhost:5000/resultat`. Les données saisies

dans le formulaire seront envoyées au serveur par l'intermédiaire de cette requête.
N.B Vous avez sans doute remarqué que la méthode à employer pour effectuer la requête HTTP n'est pas précisée dans le "@app.route('/')". Si rien n'est précisé, par défaut, c'est la méthode GET qui est utilisée.
Intéressons-nous à la fonction `resultat`, puisque c'est cette fonction qui sera exécutée côté serveur pour traiter la requête POST :

```
def resultat():
    result = request.form
    n = result['nom']
    p = result['prenom']
    return render_template("resultat.html", nom=n, prenom=p)
```

`request.form` est un dictionnaire Python qui a pour clés les attributs `name` des balises `input` du formulaire (dans notre cas les clés sont donc `nom` et `prenom`) et comme valeurs ce qui a été saisi par l'utilisateur. Si l'utilisateur saisit `Martin` et `Sophie`, le dictionnaire `request.form` sera :

```
{'nom':'Martin', 'prenom':'Sophie'}
```

Le reste du code ne devrait pas vous poser de problème.
Le template `resultat.html` utilise des paramètres `nom` et `prenom`

A faire vous même 9

Modifiez les fichiers comme suit :
`index.html`

```
<!doctype html>
<html lang="fr">
    <head>
        <meta charset="utf-8">
        <title>Le formulaire</title>
    </head>
    <body>
        <form action="http://localhost:5000/resultat" method="get">
            <label>Nom</label> : <input type="text" name="nom" />
            <label>Prénom</label> : <input type="text" name="prenom" />
            <input type="submit" value="Envoyer" />
        </form>
    </body>
</html>
```

`resultat.html` (le fichier est inchangé)

```
<!doctype html>
<html lang="fr">
    <head>
        <meta charset="utf-8">
        <title>Résultat</title>
    </head>
    <body>
        <p>Bonjour {{prenom}} {{nom}}, j'espère que vous allez bien.</p>
    </body>
</html>
```

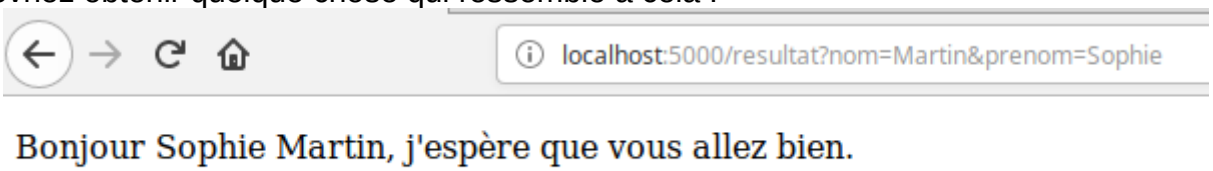
`views.py`

```
from flask import Flask, render_template, request
app = Flask(__name__)
@app.route('/')
def index():
    return render_template('index.html')
@app.route('/resultat',methods = ['GET'])
def resultat():
    result=request.args
    n = result['nom']
    p = result['prenom']
    return render_template("resultat.html", nom=n, prenom=p)
app.run(debug=True)
```

Dans `index.html`, la méthode POST a été remplacée par la méthode GET. Dans le fichier `views.py` nous avons aussi remplacé POST par GET, et on utilise `request.args` à la place de `request.form`.

A faire vous même 10

Relancez l'exécution de `views.py` et saisissez `localhost:5000` dans la barre d'adresse d'un navigateur web. Une fois la page web affichée dans votre navigateur, Saisissez "Sophie" pour le prénom et "Martin" pour le nom puis validez en cliquant sur le bouton "Envoyer". Une fois que vous avez cliqué sur le bouton "Envoyer", observez attentivement la barre d'adresse de votre navigateur. Vous devriez obtenir quelque chose qui ressemble à cela :



Vous avez dû remarquer que cette fois-ci, les informations du formulaire sont transmises au serveur par l'intermédiaire de l'URL : `localhost:5000/resultat?nom=Martin&prenom=Sophie`
Dans le cas de l'utilisation d'une méthode "POST" les données issues d'un formulaire sont envoyées au serveur sans être directement visibles, alors que dans le cas de l'utilisation d'une méthode "GET", les données sont visibles (et accessibles) puisqu'elles sont envoyées par l'intermédiaire de l'URL.

A faire vous même 11

Les données envoyées par l'intermédiaire d'une méthode "GET" peuvent être modifiées directement dans l'URL
Ouvrez votre navigateur Web et tapez dans la barre d'adresse `localhost:5000`. Une fois la page web affichée dans votre navigateur, Saisissez "Sophie" pour le prénom et "Martin" pour le nom puis validez en cliquant sur le bouton "Envoyer". Une fois que le message "Bonjour Sophie Martin, j'espère que vous allez bien." apparaît, modifier l'URL : passez
de `localhost:5000/resultat?nom=Martin&prenom=Sophie`
à `localhost:5000/resultat?nom=Martin&prenom=Jean-Pierre`
validez votre modification en appuyant sur la touche "Entrée".
Comme vous pouvez le constater, la page a bien été modifiée : "Bonjour Jean-Pierre Martin, j'espère que vous allez bien."

2.1 Une page web stylée avec Bootstrap

Les développeurs de Twitter ont, depuis longtemps, décidé de mettre en libre accès leurs feuilles de style qui sont plutôt jolie et, surtout, qui s'adapte à tout les appareils (PC, tablettes, téléphones, …). Ce « framework » s'appelle Twitter Bootstrap.

A faire vous même 12. Créer une vraie page web basé sur Bootstrap (pour les plus rapides)

3 Projet : Un site web complet avec une base de données

Par groupe de 3 élèves.

3.1 Un site web consacré aux livres de science-fiction

- Reprenez tout le travail précédent sur la base de données consacrée aux livres de science-fiction
- Créez un site web avec Flask avec :
  - une page d' accueil qui interroge et liste tous les livres de la base de données
  - une page `/ajout` qui propose un formulaire permettant d' ajouter un nouveau livre (une fois le formulaire validé, on rebascule sur la page d' accueil et le nouveau livre doit apparaître avec les autres).

3.2 Un site web de publication de posts

- Reprenez ce que nous avons fait à l' exercice 5 P. 192
- En python, créez une base de données qui reprend les relations Users et Posts
- Créez un site web avec Flask avec :
  - une page d' accueil qui affiche tous les posts avec tous leurs attributs
  - une page `/inscription` qui propose un formulaire permettant d' ajouter un nouveau User (une fois le formulaire validé, on rebascule sur la page d' accueil).
  - une page `/login` qui propose un formulaire permettant de se loguer (une fois le formulaire validé, on se dirige vers la page `ajout`).
  - une page `/ajout` (visible uniquement si l' utilisateur est logué) qui propose un formulaire permettant d' ajouter un nouveau post sachant que date/heure et auteur sont ajoutés automatiquement (une fois le formulaire validé, on rebascule sur la page d' accueil et le nouveau post doit apparaître avec les autres).

Term NSI GRILLE D'EVALUATION : Noms : Note : /10	Rien d'écrit	Ne fonctionne pas	Fonctionne mais non conforme	Fonctionne et est conforme	Fonctionne de manière optimale
Livres de SF – B. de D.	I				
Livres de SF – Web					
Publication de posts – B. de D.					
Publication de posts – Web					
Impression globale (esthétique, gestion de projet, ...)					