

Institut Supérieur Numérique

Cours Développement web avec python

Le Framework Flask

Sidi Med Hmade

C'est quoi un Framework ?

- ❖ un Framework (appelé aussi infrastructure logicielle¹) est un ensemble cohérent de composants logiciels structurels qui sert à créer les fondations ainsi que les grandes lignes de tout ou partie d'un logiciel, c'est-à-dire une architecture.
- ❖ Un Framework est une boîte à outils pour un développeur. Frame signifie cadre et work se traduit par travail.
- ❖ Un Framework contient des composants autonomes qui permettent de faciliter le développement d'un site web ou d'une application. Ces composants résolvent des problèmes souvent rencontrés par les développeurs (CRUD, sécurités, etc.).
- ❖ Ils permettent donc de gagner du temps lors du développement du site.

Framework Python

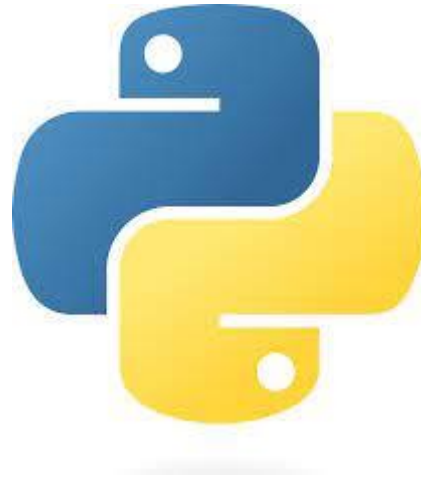


Framework Flask

- ❖ Flask est un Framework open-source de développement web en Python.
- ❖ Son but principal est d'être léger, afin de garder la souplesse de la programmation Python, associée à un système de Template.
- ❖ Il vous permet de concevoir une application web solide de manière professionnelle.
- ❖ Flask est un Framework de développement web en Python. Il en existe d'autres, le plus connu d'entre eux est Django.
- ❖ Il fournit des fonctionnalités permettant de construire des applications Web, ce qui inclut la gestion des requêtes HTTP.

Installation de Flask

❖ L'installation de Python via l'installateur officiel, nous allons prendre la version 3



❖ Installation de Flask :

- `pip install flask`



Flask

Installation de Flask

❖ Les éléments qui ont été installés :

- ✓ flask : est un framework de développement web en Python.
- ✓ Werkzeug : est le serveur web qui va répondre aux demandes des clients.
- ✓ Jinja : est un outil permettant d'insérer des éléments dynamiques dans les pages.

Première page Web avec Flask

- ❖ Créer un projet avec le script Teste.py
- ❖ Exécutez le code ci-dessous

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
    return "<h1>Hello world !</h1>"

if __name__ == "__main__":
    app.run()
```

Explication du code pas à pas

- ❖ Ce code commence par importer le module Flask :

```
from flask import Flask
```

Nous avons d'abord importé la classe Flask. Une instance de cette classe sera notre application WSGI.

Une application est dite WSGI quand elle est capable de recevoir une requête HTTP transformée en un objet Python

Explication du code pas à pas

❖ On donne ensuite un nom à l'application ici ce sera app :

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

- On instancie une application Flask.
- Une application Flask est une application web
- le 1er paramètre est le nom donné à l'application. On peut donner le nom que l'on veut.
- Ici on a utilisé l'attribut prédéfini `__name__` qui vaut `__main__`

Explication du code pas à pas

- ❖ Ensuite vient la partie cruciale : définir une page (ou route) avec flask :

`@app.route`

Permet de préciser à quelle adresse ce qui suit va s'appliquer.

Nous utilisons le décorateur `route()` pour indiquer à Flask quelle URL doit déclencher notre fonction.

Ici comme on est sur la page d'accueil, on met juste `( '/')`

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

Explication du code pas à pas

- ❖ Ensuite vient la fonction `hello()` qui va s'exécuter sur la page `'/'`. On dit simplement que quand on arrive sur la page définie par la route, on va en même temps appeler la fonction `hello()`.

Ici ça va donc afficher “Hello World” quand on arrive sur la page `“/”`.

```
def hello():
```

```
    return “<h1>Hello world !</h1>”
```

La fonction renvoie le message que nous voulons afficher dans le navigateur de l'utilisateur. Le type de contenu par défaut est HTML, donc le HTML dans la chaîne sera rendu par le navigateur.

Explication du code pas à pas

- ❖ Cette dernière partie permet juste de faire en sorte que l'application se lance quand on lance le code dans la console ou le terminal.

```
if __name__ == "__main__":  
    app.run()
```

- ❖ Pour changer le port :

```
if __name__ == "__main__":  
    app.run(port=3000)
```

l'application web est lancée : un serveur web est instancié et l'application web est déployée dessus afin de répondre aux requêtes de clients web.

Explication du code pas à pas

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
    return "Hello world !"

if __name__ == "__main__":
    app.run()
```

```
from flask import Flask

app = Flask('teste')

@app.route('/')
def hello():
    return "Hello world !"

if app.name == "teste":
    app.run()
```

Debug Mode

❖ Pour gérer les bugs :

```
if __name__ == "__main__":  
    app.run(debug = True)
```

La commande flask **run** peut faire plus que simplement démarrer le serveur de développement. En activant le mode débogage, le serveur se rechargera automatiquement si le code change et affichera un débogueur interactif dans le navigateur si une erreur se produit lors d'une requête.

Routing

- ❖ **app** est une application Flask.
- ❖ **route** : les applications Web modernes utilisent des URL significatives pour aider les utilisateurs, en développement Web, on appelle route une URL ou un ensemble d'URLs conduisant à l'exécution d'une fonction donnée. Dans Flask, les routes sont déclarées via le décorateur **app.route**.

Routing

- ❖ Utilisez le décorateur `route()` pour lier une fonction à une URL.

```
@app.route('/')  
def index():  
    return 'Index Page'  
  
@app.route('/hello')  
def hello():  
    return 'Hello, World'
```

Routing

- ❖ Une route peut être paramétrée, auquel cas le paramètre sera passé à la fonction vue :

```
@app.route('/ind/<name>')  
def teste(name):  
    return 'hello ' + str(name)
```

Génération d'URL

- ❖ Les routes permettent à Flask de trouver la vue correspondant à une URL, mais également de faire l'inverse, à savoir de reconstruire l'URL d'une vue donnée.
- ❖ La fonction `flask.url_for` prend en paramètre le nom d'une vue (le nom de la fonction, dans une chaîne de caractères), avec ses éventuels paramètres, et retourne l'URL correspondante.

Génération d'URL

❖ avec les routes des exemples précédents :

```
url_for('index')           # → "/"  
url_for('teste', name="Sidi") # → "/ind/Sidi"
```

❖ **url_for** rend le code plus facile à maintenir : si on décide de changer l'URL d'une vue, il suffit de changer le paramètre de app.route, et toutes les autres vues contenant des liens vers elle s'adapteront automatiquement.

Template Jinja2

- ❖ Un Template, ou modèle, est un fichier dont certaines parties seront remplacées à l'exécution.
- ❖ Voici un exemple :

`<p>Hello {{name}}</p>`

où la partie entre doubles-accolades sera remplacée par le contenu de la variable name,

par exemple : `<p>Hello world</p>`

Template en Flask

- ❖ Flask utilise le système de templates Jinja2, qui permet de générer n'importe quel format textuel (HTML, CSS...).
- ❖ Les templates sont stockés dans un répertoire templates, situé dans le même répertoire que le fichier Python définissant l'application.
- ❖ Pour appliquer un Template, il suffit d'appeler la fonction **flask.render_template** en lui passant + le nom du Template (relative au répertoire templates), et + la liste des variables utilisées par le Template :

Template en Flask

- ❖ Vous pouvez utiliser la méthode `render_template()` : Tout ce que vous avez à faire est de fournir le nom du Template et les variables que vous souhaitez transmettre au moteur du Template en tant qu'arguments.

```
@app.route("/bon")
def bonjour():
    return render_template("index.html")

@app.route("/hello/<n>")
def hello(n):
    return render_template("hello.html", name=n)
```

Template en Flask

❖ Voici un exemple simple de rendu d'un modèle :

```
from flask import Flask, render_template  
app = Flask(__name__)  
@app.route("/")  
def hello(name= 'Med'):  
    return render_template("home.html", message="Hello World!" + name)
```

Substitution

- ❖ Dans une Template Jinja2, les doubles-accolades `{{ }}` servent à indiquer une substitution.
- ❖ Elles peuvent contenir un nom de variable, mais également des expressions plus complexes, par exemple `{{request.method}}` ou `{{row[0]}}`.
- ❖ Certaines variables sont transmises automatiquement aux templates, notamment l'objet requête `request`.

Structures de contrôle

- ❖ Jinja2 offre des structures de contrôles (condition, boucle) similaires à celles de Python.
- ❖ Elles sont encadrées par les symboles {% %}.
- ❖ Exemple :

```
@app.route('/')  
def acc():  
    L=['Google', 'Youtube', 'coursera']  
    return render_template("home.html", elements= L)
```

Structures de contrôle

❖ Exemple :

```
<body>
  {% if elements %}
    <ul>
      {% for e in elements %}
        <li>{{e}}</li></h1>
      {% endfor %}
    </ul>
  {% else %}
    <p>Aucun élément</p>
  {% endif %}
</body>
```

Ressources statiques

- ❖ Une application Web ne se limite pas aux contenus HTML générés par les vues, on a également besoin de ressources statiques (CSS, images...).
- ❖ Dans Flask, il est possible de stocker des fichiers dans un répertoire static, situé dans le même répertoire que le fichier Python définissant l'application.
- ❖ L'URL de ces fichiers est donnée par la fonction `url_for` :

`url_for("static", filename="nom_du_fichier.css")`

```
<link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='style.css') }}">
```

Formulaires

❖ Routes et méthodes :

Il est possible de spécifier la/les méthode(s) autorisée(s) pour une vue :

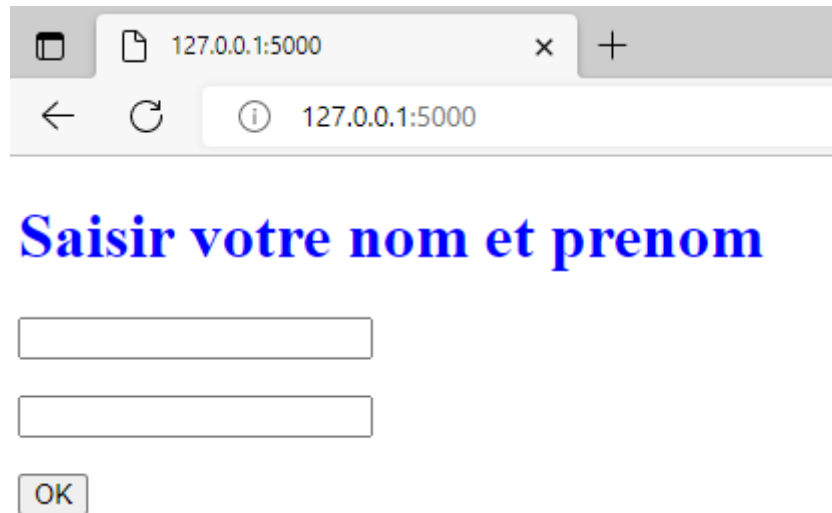
```
@app.route("/test", methods=['GET', 'POST'])
def teste():
    if request.method == 'GET':
        # ...
    else:
        # ...
```

Formulaires

❖ Récupération des paramètres :

- ✓ **request.args** est un dictionnaire contenant les paramètres d'URL.
- ✓ **request.form** est un dictionnaire contenant les données de formulaires envoyées par POST

❖ Exemple :



The screenshot shows a web browser window with the address bar displaying "127.0.0.1:5000". The page content includes the text "Saisir votre nom et prenom" in blue, followed by two empty text input fields and an "OK" button.

Bienvenue Med Ali

[Retourner à la page d'accueil](#)

Formulaires

❖ Récupération des paramètres :

```
@app.route('/', methods=['POST'])
def teste_form():
    nom = request.form['Nom']
    prenom = request.form['Prenom']
    text = nom + ' ' + prenom
    return render_template("bienvenue.html", message=text)
```

❖ La page bienvenue.html

```
<h1>Bienvenue {{message}}</h1>
<a href="{{url_for('indx')}}">Cliquer ici pour retourner vers la page</a>
```

Formulaires

❖ Récupération des paramètres :

```
<!DOCTYPE html>
<html lang="en">
<head>
  <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='style.css') }}">
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
  <h1>Saisir votre nom et prenom</h1>
  <form method="post" action="{{url_for('testForm')}}">
    <input type="text" name="nom"> <br> <br>
    <input type="text" name="prenom"> <br> <br>
    <input type="submit" name="Ok" value="OK">
  </form>
</body>
</html>
```

Redirection

- ❖ Pour rediriger le client vers une autre URL, on peut utiliser la fonction **flask.redirect**
- ❖ Exemple :

```
return redirect(url_for("root"))
```

Redirection

❖ Exemple :

```
@app.route('/login', methods=['POST', 'GET'])  
def login():  
    if request.method == 'POST' and request.form['username'] == 'admin':  
        return redirect(url_for('success'))  
    return redirect(url_for('index'))
```

Sessions

- ❖ En plus de l'objet requête, il existe un second objet appelé session qui permet de stocker des informations spécifiques à un utilisateur d'une requête à l'autre.
- ❖ Cet objet implémente les cookies de manière cryptographique. Cela signifie que l'utilisateur peut regarder le contenu de votre cookie mais ne peut pas le modifier, sauf s'il connaît la clé secrète utilisée pour la signature.
- ❖ Pour utiliser les sessions, vous devez définir une clé secrète.

Créer une session

- ❖ Pour créer une session, on a besoin de deux choses:
 - ✓ Importer session depuis flask
 - ✓ Créer une clef de sécurité pour s'assurer qu'on soit le seul à pouvoir lire et écrire nos cookies
- ❖ Créer la clef de sécurité se fait tout en haut de notre programme, juste après avoir créé notre application avec la variable `app.secret_key`.

```
from flask import Flask, session  
  
app = Flask("toto")  
app.secret_key = "MOT_de_passe_Pour_les_COOKIES"
```

Utiliser la session

- ❖ La session est un dictionnaire dans lequel on va pouvoir créer de nouvelles clefs pour y stocker de nouvelles valeurs.
- ❖ Créer une nouvelle clef : Par exemple, ici je crée une variable util qui se souviendra du nom de l'utilisateur.

```
session['util']='toto'
```

- ❖ Accéder à une valeur : Pour accéder à la valeur stockée, il me suffit d'écrire session['util']

```
session['util']
```

- ❖ Supprimer tous les cookies : Pour supprimer toutes les valeurs du dictionnaire on peut utiliser la fonction **clear**.

```
session.clear()
```

Flask SQLAlchemy

- ❖ L'utilisation de SQL brut dans l'application Flask Web pour effectuer des opérations CRUD sur la base de données peut être fastidieuse.
- ❖ SQLAlchemy est un ORM (Object-Relational Mapping) permettant de synchroniser vos classes avec des tables en base de données relationnelle (basée sur SQL).
- ❖ L'étape la plus importante, quand on utilise un ORM, c'est de réaliser le « Mapping », c'est à dire l'association entre les éléments de vos classes et ceux de vos tables en base de données.
- ❖ L' ORM permet d'effectuer des opérations CRUD sans écrire d'instructions SQL brutes.

Flask SQLAlchemy

- ❖ Étape 1 : Installez l'extension Flask-SQLAlchemy

```
pip install flask-sqlalchemy
```

- ❖ Étape 2 : Vous devez importer la classe SQLAlchemy à partir de ce module

```
from flask_sqlalchemy import SQLAlchemy
```

- ❖ Étape 3 : Créez maintenant un objet d'application Flask et définissez l'URI de la base de données à utiliser.

```
app = Flask(__name__)  
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///users.sqlite3'
```

Flask SQLAlchemy

- ❖ Étape 4 : utilisez l'objet application comme paramètre pour créer un objet de classe SQLAlchemy.

```
db = SQLAlchemy(app)
```

- ❖ Étape 5 : créer la classe model

```
class User(db.Model):  
    id = db.Column(db.Integer, primary_key=True)  
    username = db.Column(db.String(20), unique=True, nullable=False)
```

Flask SQLAlchemy

- ❖ Étape 6 : Pour créer/utiliser la base de données mentionnée dans l'URI, exécutez la méthode *create_all()*

```
db.create_all()
```

CRUD

- ❖ L'objet Session de SQLAlchemy gère toutes les opérations de persistance des objets ORM.
- ❖ La méthode de session suivante effectue des opérations CRUD :
 - Insérer dans la table de mapping :

```
db.session.add (model object)
```

- Supprimer un enregistrement :

```
db.session.delete (model object)
```

CRUD

- Récupérer tous les enregistrements (correspondant aux requêtes SELECT) de la table.

```
model.query.all ()
```

- Vous pouvez appliquer des filtres sur les enregistrements récupérés par `model.query.all ()`,

- ✓ Pour récupérer des enregistrements pour `city = 'Tokyo'` dans les tables d'étudiants, utilisez l'instruction suivante :

```
etudiants.query.filter_by(city = 'NKC').all()
```

CRUD

```
from flask import Flask, render_template
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///users.sqlite3'

db = SQLAlchemy(app)

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(20), nullable=False)

@app.route('/')
def affichage():
    return render_template('view.html', users=User.query.all())

with app.app_context():
    db.create_all()
    db.session.add(User(username="sidi"))
    db.session.commit()
    app.run(port=3333, debug=True)
```

CRUD

```
<body>
  {% for user in users %}
    {{ user.id }}
    {{ user.username }} <br>

  {% endfor %}
</body>
```