UAA12 Flask - TP1: Prise en main

Le but est de prendre en main flask en suivant le tutoriel [12] Lycée des Flandres, Formulaire : utilisation de Flask
https://qkzk.xyz/docs/nsi/cours_premiere/ihm_web/flask

1 Test1 : 1ère page views.py

1. Dans le dossier **2_uaa12_python_flask**, crée uns un dossier **tp1** et un sous dossier **test1_view**.
2. Ouvre un terminal linux, active l'environnement de programmation virtuel et lance flask

Code dans un terminal linux:

```
startflask
```

Résultat dans le terminal :

```
(env) t5info@localhost:$
```

Remarque

startflask est un alias (un raccourci de commande) qui va dans le dossier public_html, active l'environnement virtuel et installe flask :

```
cd public_html
source env/bin/activate
pip install flask
```

3. Il n'est plus nécessaire d'installer flask les prochaines fois. Tu peux directement te déplacer par le gestionnaire de fichier dans le dossier **test1_view**, ouvrir un terminal et activer l'environnement de programmation virtuel

Code dans un terminal linux:

```
startenv
```

4. Télécharge du drive le dossier **flask_template**, et copie dans **test1_view** le fichier **view.py**. Ouvre le fichier-le qui suit et modifie la ligne contenant l'instruction return. (**[12] A faire vous-même 3)**

Code de view.py

```
from flask import *
app = Flask(__name__)

@app.route('/')
def index():
   return "<p> Tout fonctionne parfaitement </p>"

if __name__ == '__main__':
   app.run ("0.0.0.0", "5010",debug=True)
```

5. Exécute ton programme (il lance un serveur flask sur ton pc)

Code dans le terminal de l'env virtuel:

```
(env) t5info@localhost:$ python ./view.py
```

Résultat dans le terminal :

```
* Serving Flask app 'view' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
* Running on http://127.0.0.1:5010/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 524-242-581
```

6. Ouvre ton navigateur et tape dans la barre d'adresse l'URL suivante.

   Accède au serveur en suivant les consignes (**[12] A faire vous-même 3)**

URL du navigateur:

```
localhost:5010
```

Tu dois voir apparaître le message suivant s'afficher dans le navigateur :

```
Tout fonctionne parfaitement
```

Si tu regardes le code source à partir de ton navigateur (click droit -> code source), tu dois voir le contenu de la page html :

```
"<p> Tout fonctionne parfaitement </p>
```

Explications

"**localhost**" indique au navigateur que le serveur web se trouve sur le même ordinateur que lui (on parle de machine locale). Dans un cas normal, la barre d'adresse devrait être renseignée avec l'adresse du serveur web.
"**:5010**" indique le port, il doit suivre le "localhost".
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 5010. En ouvrant un navigateur web et en tapant "localhost:5010", nous faisons une requête HTTP.
Le serveur web fourni avec Flask répond à cette requête HTTP en envoyant une page web contenant uniquement Tout fonctionne parfaitement .

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 systématiquement nécessaire.

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

Nous utilisons ici un décorateur. La fonction ("index") qui suit ce décorateur, 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 saisit dans la barre d'adresse "localhost:5010/" (ou simplement "localhost:5010") Nous verrons ci-dessous un exemple avec une URL autre que '/'.

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

En cas de requête HTTP d'un client avec l'URL "/", la fonction index() est exécutée et le serveur renvoie vers le client une page HTML contenant uniquement la ligne Tout fonctionne parfaitement (instruction return de la fonction).

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

Cette ligne permet de lancer le serveur, elle sera systématiquement présente. Le paramètre debug=True permet d'avoir des informations de debugage dans le navigateur.
Ce n'est souhaitable que durant le *développement* d'un site web.

7. Ouvre un terminal, et lis l'adresse IP de ton pc.

Code dans un terminal linux:

```
hostname -I
```

8. Donne-là à ton voisin pour qu'il accède à ta page en tapant dans l'URL de son navigateur (tu peux changer le message de ton fichier view.py que voit ton voisin):

```
adresse_ip:5010
```

9. En fin de séance désactive l'environnement virtuel

Code dans le terminal de l'env virtuel:

```
(env) t5info@localhost:$ stopflask
```

2 Test2 : Afficher l'heure

Dans **tp1**, crée un dossier **test2_heure**
L'objectif est d'afficher l'heure dans une page html

page dans le navigateur (via le serveur)

```
Bienvenue sur mon site
Il est 18h 23m 15s
```

1. Pour cela, déplace-toi dans le dossier **0_uaa12_intro**, copie dans **test2_heure** hello.py et view.py.
2. Dans **test2_heure**, essaie de comprendre le contenu de **hello.py** et exécute-le avec idle (en python seul).
3. Essaie de comprendre le contenue de ***view.py***
4. Ouvre un terminal et active l'environnement de programmation virtuel

Code dans un terminal linux:

```
startenv
```

Exécute le fichier **view.py**.

Code dans le terminal de l'env virtuel:

```
(env) t5info@localhost:$ python ./view.py
```

Accède au serveur par le navigateur

URL du navigateur:

```
localhost:5010
```

Que se passe-t-il quand tu recharges la page ? Comment évolue le code de la page html ?
Sur le test1, notre site est statique : la page reste identique, quelles que soient les actions des visiteurs.

Sur le test2, notre site est dynamique :

- 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

3 Test3 : 2ème page about

1. Crée un dossier **test3_about**, copie le fichier view.py et modifie-le (**[12] A faire vous-même 4)**

Code de view.py

```
from flask import *
app = Flask(__name__)

@app.route('/')
def index():
   return "<p> Tout fonctionne parfaitement </p>"

@app.route('/about')
def about():
  return "<p>Une autre page</p>"

if __name__ == '__main__':
   app.run ("0.0.0.0", "5010",debug=True)
```

Après avoir exécuté le programme ci-dessus, saisis "**localhost:5010/about**" dans la barre d'adresse de votre navigateur.
Comme tu peux le constater, le serveur nous renvoie dans ce cas une autre page. Évidemment l'URL racine ("**/**") reste disponible, tu peux passer d'une page à l'autre en modifiant l'URL dans la barre d'adresse :

URL du navigateur:

```
localhost:5010
```

```
localhost:5010/about
```

2. Modifie la fonction index dans view.py et vérifie que les liens marchent correctement :

Code dans view.py:

```
@app.route('/')
def index():
    message = """
            <h1> Tout marche parfaitement </h1>
             <p> <a href ="https://pythonbasics.org"> lien vers pythonbasics.org </a> </p>
             <p> <a href ="/about"> lien vers la page about </a> </p>
        """
    return message
```

3. Exercice : dans la fonction about(), rajoute un lien de retour vers la page d'index.

4 Test4 : Exercice liens entre 2 pages html

Reprends les 2 pages test1p1.html et test2p2.html de l'exercice d'introduction test1. Crée un fichier **test4_exo** et écris un fichier view.py afin qu'elles soient consultables depuis le serveur flask.

5 Test5 : Page index.html dans le dossier template

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.

1. Copie **TOUT** le dossier flask_template et renomme-le **test5**. ([12] A faire vous-même 5 & 6)

Ce dossier contient l'arborescence ci-dessous. Attention, pour chaque nouvel exercice, tu devras copier tous les dossiers/fichiers de flask_template et toujours respecter l'arborescence suivante :

Arborescence flask dans test5

```
-> view.py
-> templates
        -> index.html et autres pages htm
-> static
        -> css      -> style.css
        -> images   -> images ...
        -> js       -> programmes javascript
```

2. Modifie la page d'index afin d'afficher le texte suivant :

index.html dans le navigateur

```
Mon super site
Tout fonctionne parfaitement
```

3. Observe le code modifié dans le view.py

Contenu de view.py modifié

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

Exécute le fichier view.py et accède au serveur par le navigateur.

URL du navigateur:

```
localhost:5010
```

Le serveur renvoie maintenant au client la page html correspondant au fichier **index.html** qui a été créé dans le dossier **templates**. Attention, **la fonction render_template s'attend à trouver le fichier html dans le dossier templates. Les fichiers html devront donc systématiquement se trouver dans ce dossier et le nom du dossier ne doit pas être changé.**

6 Test6 : Template html avec des variables Jinja

L'objectif est d'afficher l'heure dans la page index html

index.html dans le navigateur

```
Bienvenue sur mon site
Il est 18h 23m 15s
```

1. Copie TOUT le dossier flask_template et renomme-le **test6_var_jinja** et modifie le fichier view.py comme ci-dessous (**[12] A faire vous-même 7 & 8)**

Code de view.py

```
from flask import *
from time import *

app = Flask(__name__)

@app.route('/')
def index():
    tactuel=localtime(time())
    h_py= tactuel.tm_hour
    m_py = tactuel.tm_min
    s_py = tactuel.tm_sec
    return render_template("index.html", heure_jin=h_py, minute_jin=m_py,
                seconde_jin =s_py)

if __name__ == '__main__':
   app.run ("0.0.0.0", "5010",debug=True)
```

Les 3 variables h, m, s récupèrent l'heure actuelle.

La fonction render_template contient 3 paramètres de plus par rapport à l'exemple précédent: les paramètres heure, minute, seconde.
Nous allons retrouver ces 3 paramètres dans le fichier HTML qui affichera l'heure à l'écran comme au test2.

2. Rajoute dans index.html les variables passées par la fonction render_template()

Code de index.html

```
<body>
    <h1>Bienvenue sur mon site</h1>
    <p>Il est {{heure_jin}} h {{minute_jin}} minutes {{seconde_jin}}  secondes </p>
</body>
```

3. Ouvre ta page dans le navigateur

Explications

Attention, il est bien important de comprendre que la page html envoyée par le serveur au client ne contient plus les paramètres {{heure}}, {{minute}} et {{seconde}}. Au moment de créer la page, le serveur remplace ces paramètres par les valeurs passées en paramètres de la fonction render_template
Le fichier "index.html" ne contient donc pas du html (même si cela ressemble beaucoup à du HTML), car les paramètres {{heure}}, {{minute}} et {{seconde}} n'existent pas en HTML. Le fichier "index.html" contient en fait un langage de template nommé **Jinja**.
Jinja ajoute beaucoup de fonctionnalités par rapport au HTML (notamment les paramètres entourés d'une double accolade comme {{heure}}).
Si vous utilisez Jinja seul (sans un framework comme Flask), les paramètres ne seront pas remplacés et votre navigateur affichera :
"Le serveur fonctionne parfaitement, il est {{heure}} h {{minute}} minutes et {{seconde}} secondes".
C'est donc Flask qui transforme ces {{ }} en du code HTML compréhensible par le navigateur.

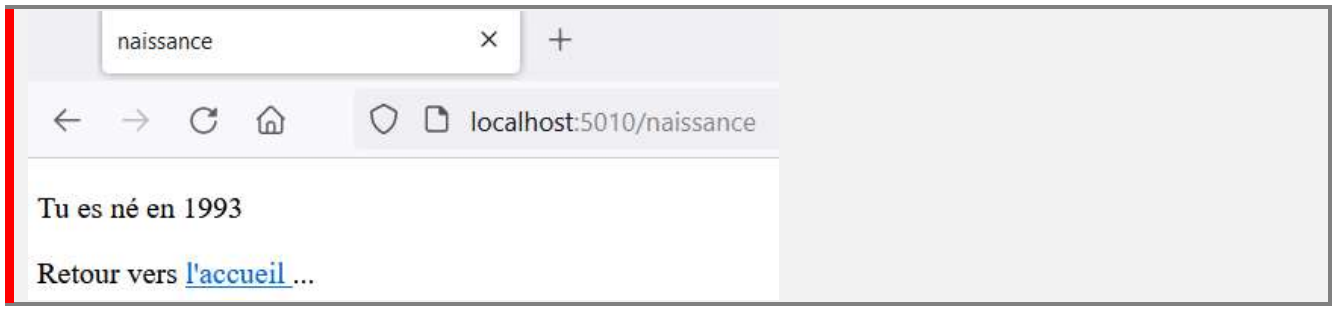
7 Test7 : exercice récapitulatif

1. Copie TOUT le dossier flask_template et renomme-le **test7_exo_recap**.

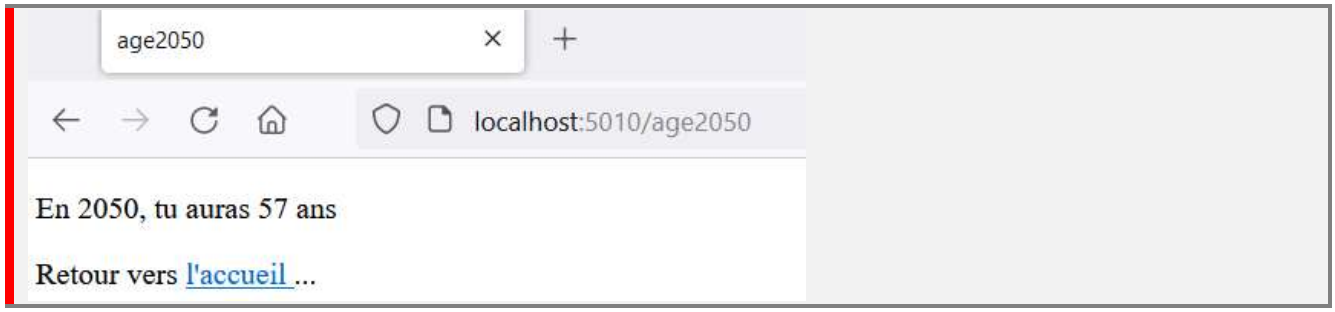
L'objectif est de créer les 3 pages html ci-dessous: les pages **index.html, naissance.html, age2050.html** sont appelées par le fichier view.py.
Les variables du **nom, prénom, âge, date de naissance, âge en 2050** sont définies dans le fichier view.py
Les **3 liens** permettent de naviguer vers les différentes pages

index.html dans le navigateur (via le serveur)

naissance.html dans le navigateur (via le serveur)



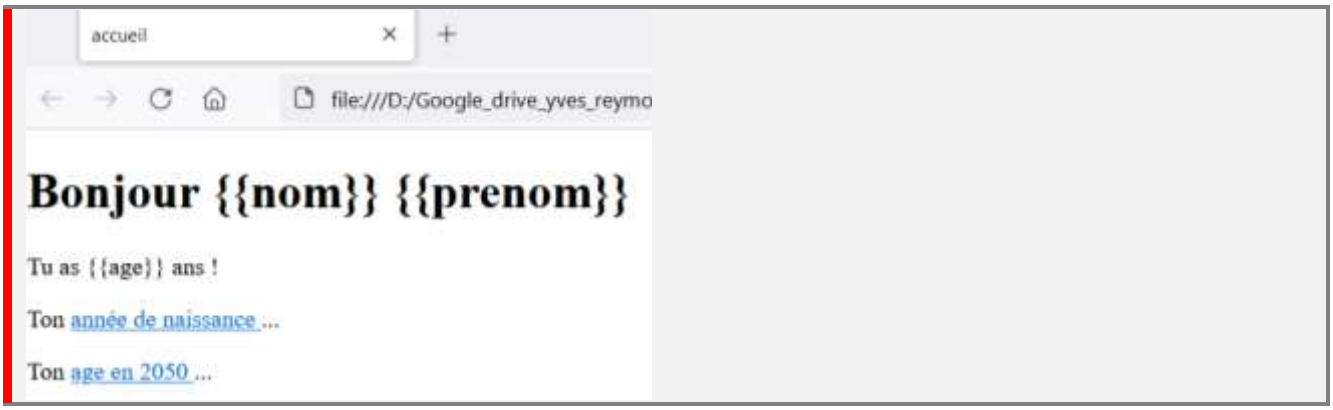
age2050.html dans le navigateur (via le serveur)



2. Démarche

1) **Ecris les 3 fichiers html** incluant les variables **jinja** et les liens vers les **URL** des pages. Ouvre-les directement avec le navigateur (sans démarrer le serveur).

Exemple : index.html directement ouvert dans le navigateur



Attention, les liens href doivent pointer sur les URL que tu vas définir avec les instructions @app.route('/') dans le fichier view.py et non directement vers les fichiers html.

2) **Ecris le fichier view.py** en mettant autant de **route()** qu'il y a de pages html à charger, ici 3.
Tu peux visualiser le contenu des variables dans le terminal en rajoutant des print() et en commentant les fonctions render_template().
Teste tes pages en lançant le serveur.

Exemple: code de la fonction index()

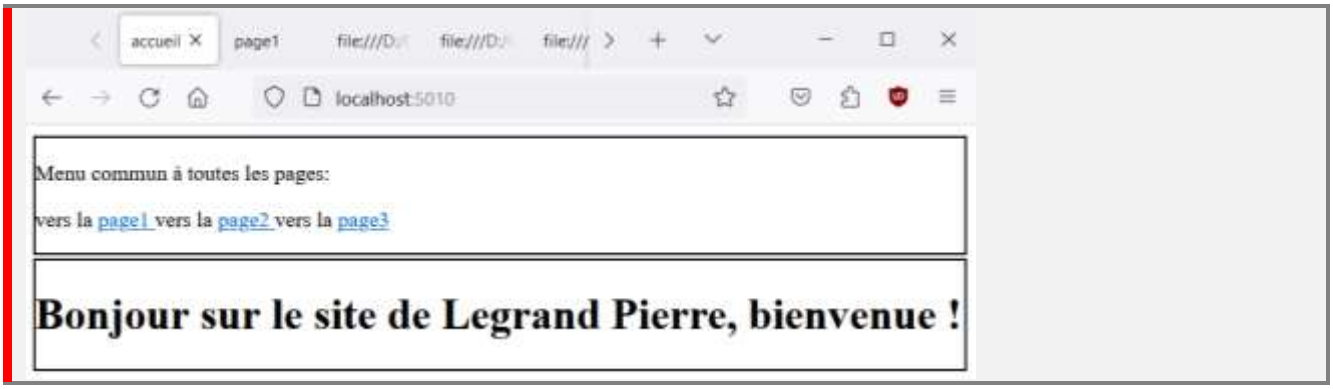
```
@app.route('/')
def index():
    n_py='Legrand'
    p_py ='Alexandre'
    print("debug - n / p: ",n_py,p_py)
```

3) **Complète le fichier view.py** en écrivant les instructions « return **render_template()** » afin de charger les 3 pages html avec les variables jinja.
Tu peux alors vérifier le fonctionnement de tes 3 pages.

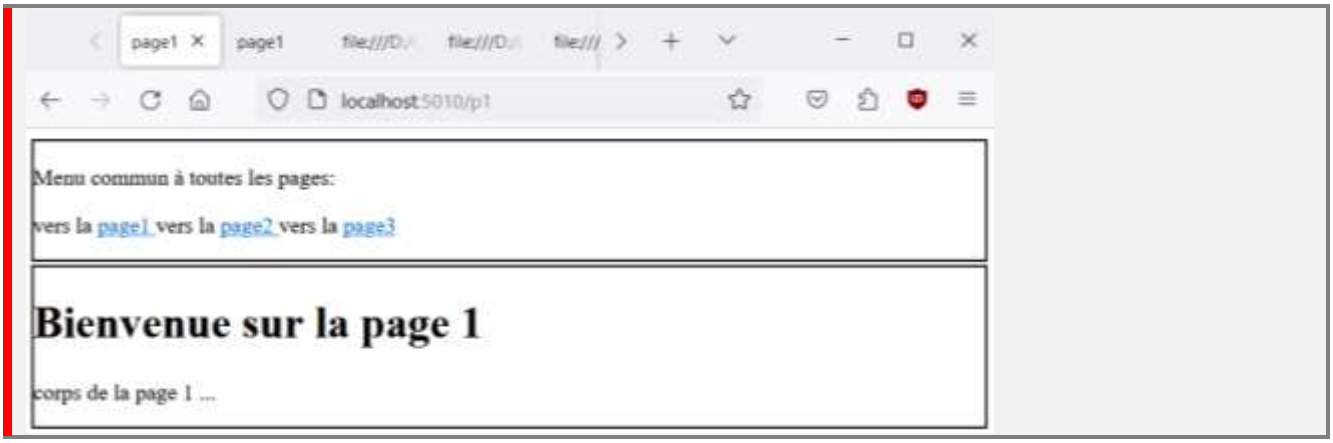
8 Test 8 : 1ère étape d'un site de 3 pages

1. Copie TOUT le dossier flask_template et renomme-le **test8_site_3p**.
2. Crée une **page d'accueil et 3 pages** contenant une partie commune (le menu permettant de naviguer d'une page à l'autre), et une partie propre à chacune (le message de bienvenue).

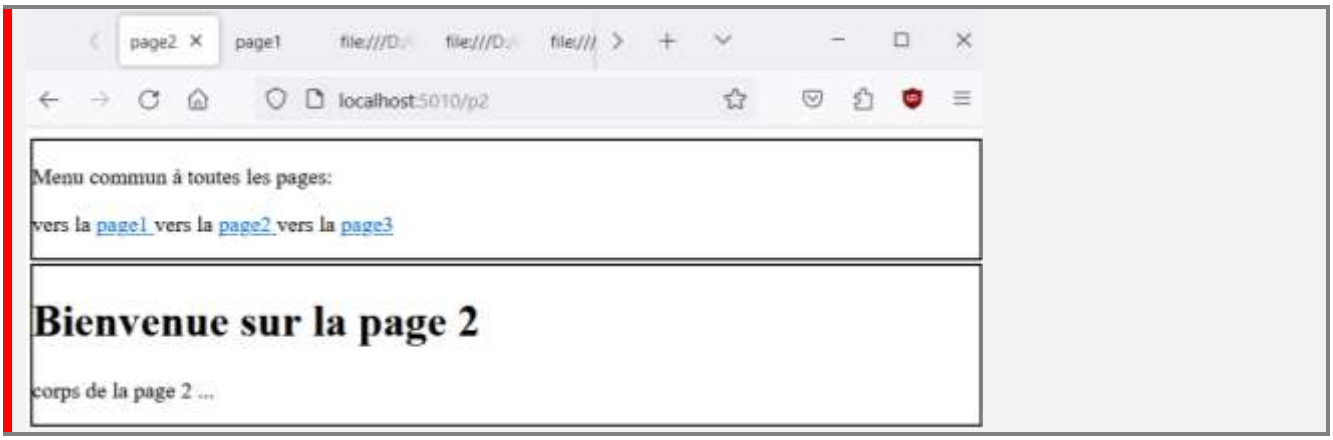
index.html dans le navigateur



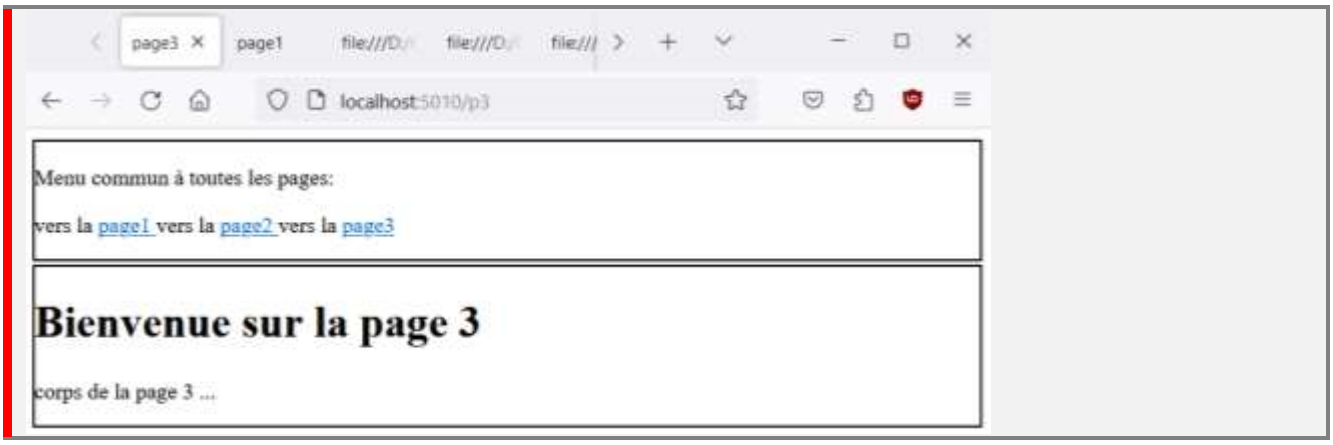
page1.html dans le navigateur



page2.html dans le navigateur



page3.html dans le navigateur



3. Le **menu** est défini dans un fichier à part. Il ne contient que le code ci-dessous (pas de <html>, ni de <head>, ni de <body>)

Code de menu.html

```
<div>
    <p> Menu commun à toutes les pages: </p>
    <p> vers la <a href="/p1"> page1 </a> vers  la <a href="/p2"> page2 </a> vers la <a href="/p3"> page3 </a> </p>
</div>
```

Il est appelé par chaque page par l'instruction jinja **include** :

```
{% include 'menu.html' %}
```