

Le moteur de template Jinja

Table des matières

I. Créez votre premier template Jinja	3
A. La création de notre premier template	3
B. L'ajout des variables.....	4
II. Exercice : Quiz	7
III. Apprenez à utiliser l'héritage, les blocs et les boucles	8
A. L'héritage en Jinja	8
B. L'ajout du CSS.....	11
C. Les boucles en Jinja	11
D. Les conditions en Jinja	14
IV. Exercice : Quiz	16
V. Essentiel	17
VI. Auto-évaluation	18
A. Exercice	18
B. Test.....	18
Solutions des exercices	19

I. Créez votre premier template Jinja

Prérequis

- Une connaissance pratique du langage de programmation Python,
- Une connaissance du *framework* Flask,
- Les base du HTML.

Contexte

Le but de ce cours est d'apprendre les bases du *template* Jinja2 en utilisant le *framework* Flask.

Python est un langage de programmation qui peut être utilisé afin de développer des sites web grâce à ses *frameworks* puissants, tels que Django et Flask.

Jinja2 est un moteur de *template* (*template engine*) qui permet d'importer d'autres *templates* et d'utiliser des conditions « *if* », des boucles « *for* », des variables, etc.

Afin de pouvoir suivre ce cours, il est indispensable d'avoir les prérequis mentionnés.

A. La création de notre premier template

Fondamental

Pour créer un *template* en Python en utilisant le *framework* Flask, il faut tout d'abord commencer par créer un dossier `/templates`. Ensuite nous créons une page HTML qu'on appellera `homePage.html` et qui va être notre *template*, qui sera généré à partir de la partie *back-end* de notre projet :

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>Document</title>
8 </head>
9 <body>
10  <p>Hello world</p>
11 </body>
12 </html>
```

Pour générer le fichier HTML qu'on vient de créer, il suffit d'importer et d'appeler la fonction `render_template`, cette fonction prend en paramètre le nom du *template* et la liste des variables qui sera utilisée par le *template*.

Méthode

```
1 @skills_app.route("/")
2 def homepage():
3     return render_template('homePage.html')
```

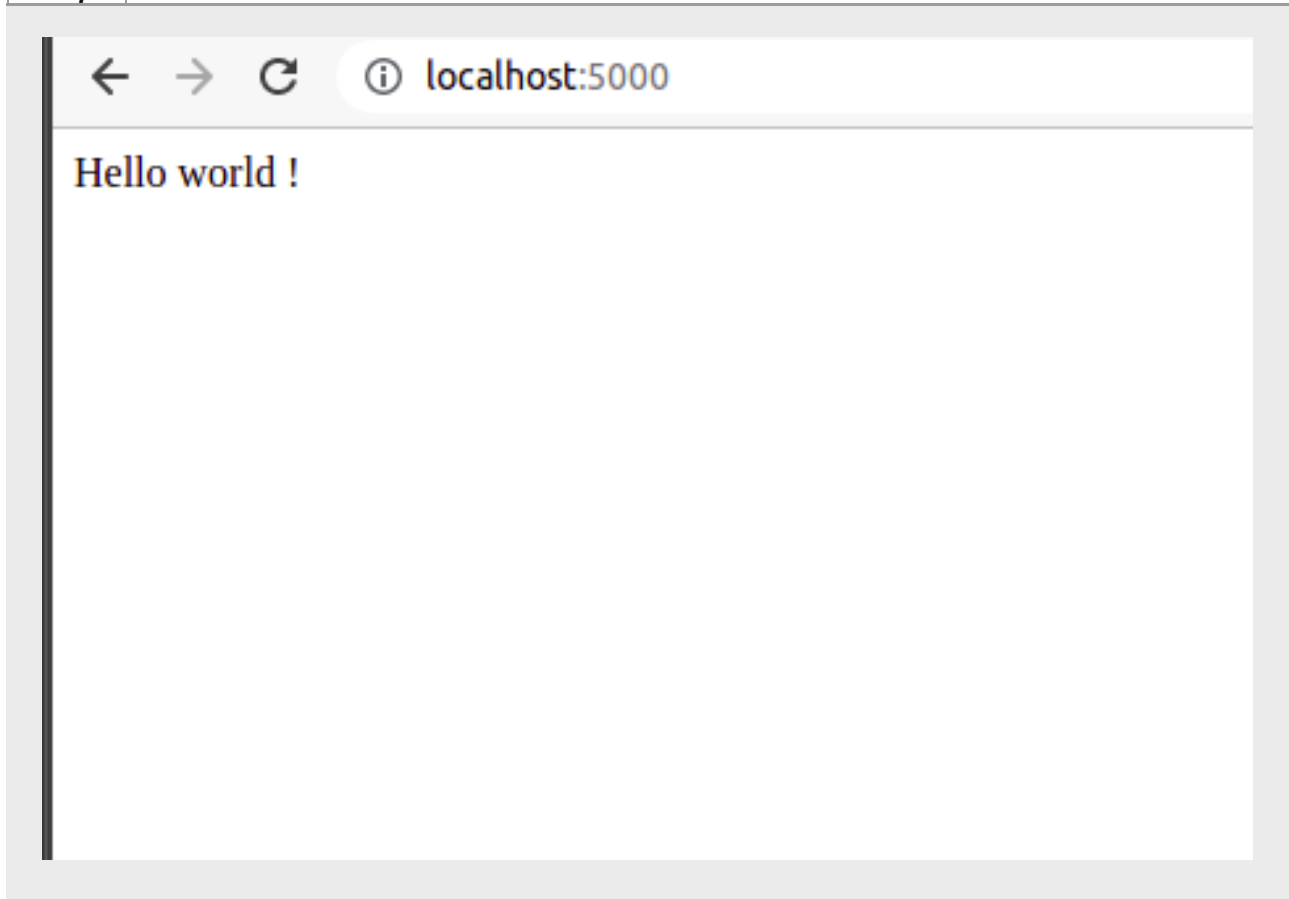
La route `/` va donc représenter notre page d'accueil, c'est-à-dire qu'elle correspondra à l'URL : `localhost:5000/`.

Fondamental

Une fois que notre programme est exécuté, Flask va aller chercher dans le dossier `/templates` le fichier `homePage.html` et il affichera notre page HTML.

Sur le port 5000 de notre `localhost` (127.0.0.1) nous obtenons le résultat « *Hello world !* ».

Exemple



B. L'ajout des variables

Fondamental

Les *templates* sur Flask nous donnent aussi la possibilité d'ajouter une liste des variables qui peut être utilisée par le *template*. Si nous prenons notre exemple précédent, nous allons essayer cette fois d'ajouter un titre (dans le contenu de la balise `<title> </title>`) à notre page de façon dynamique, en utilisant les variables.

Nous allons commencer par modifier le code de la partie *back-end* en ajoutant la variable `myTitle` à la fonction `render_template`.

Méthode

Dans la racine de notre projet, nous ajoutons dans le fichier **server.py** :

```
1 @skills_app.route("/")
2 def homepage():
3     return render_template('homePage.html', myTitle='mon nouveau title')
```

Fondamental

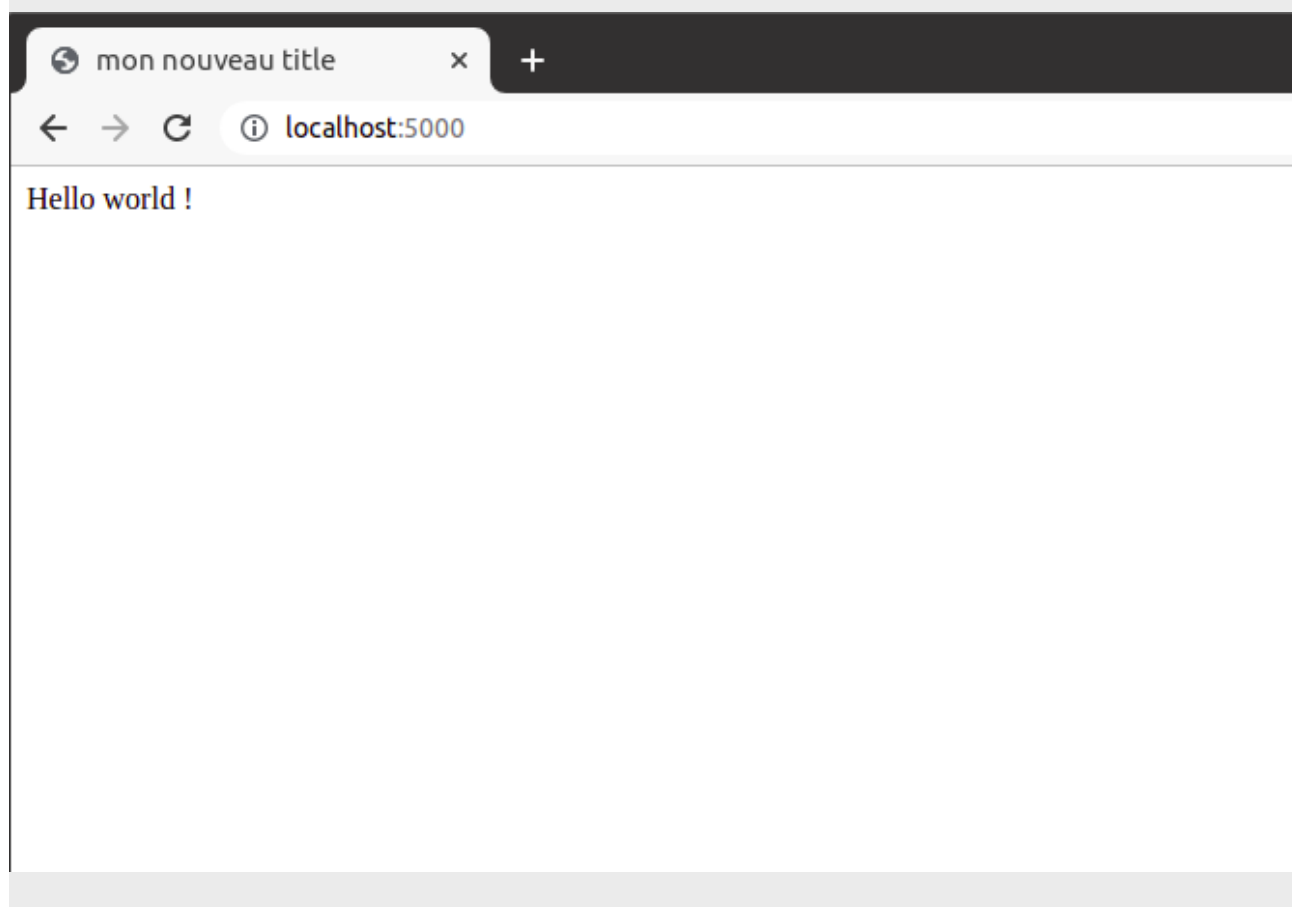
Ensuite, pour afficher dynamiquement le titre de notre page, nous allons mettre la variable `myTitle` entre deux accolades dans notre `template homePage.html` dans le contenu de la balise `<title> </title>`.

Méthode

Dans :

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>{{myTitle}}</title>
8 </head>
9 <body>
10   Hello world !
11 </body>
12 </html>
```

Dans l'onglet du navigateur, on voit le titre « *mon nouveau titre* » qui s'affiche :

Exemple

Fondamental

Dès à présent, nous avons donc la possibilité de modifier le titre de notre page de façon dynamique, par conséquent nous venons juste de découvrir l'un des premiers objectifs principaux des *templates*, qui est le fait de pouvoir créer un seul *template* pour une ou plusieurs pages.

Par exemple, supposons que l'on souhaite créer une dizaine de pages web qui contiennent à peu près les mêmes éléments mais avec des paragraphes différents, dans ce cas-là, au lieu d'ajouter à notre projet 10 pages HTML, il suffit juste de créer un seul *template* et de changer dynamiquement le contenu des paragraphes concernés.

Notez bien que l'on peut ajouter dans la fonction `render_template` autant de variables que l'on souhaite.

Méthode

```
1 server.py :
2
3 @skills_app.route("/")
4 def homepage():
5     return render_template('homePage.html', myTitle='mon nouveau titre', myVariable2='ma
    deuxième variable', myVariable3='ma troisième variable')
```

Fondamental

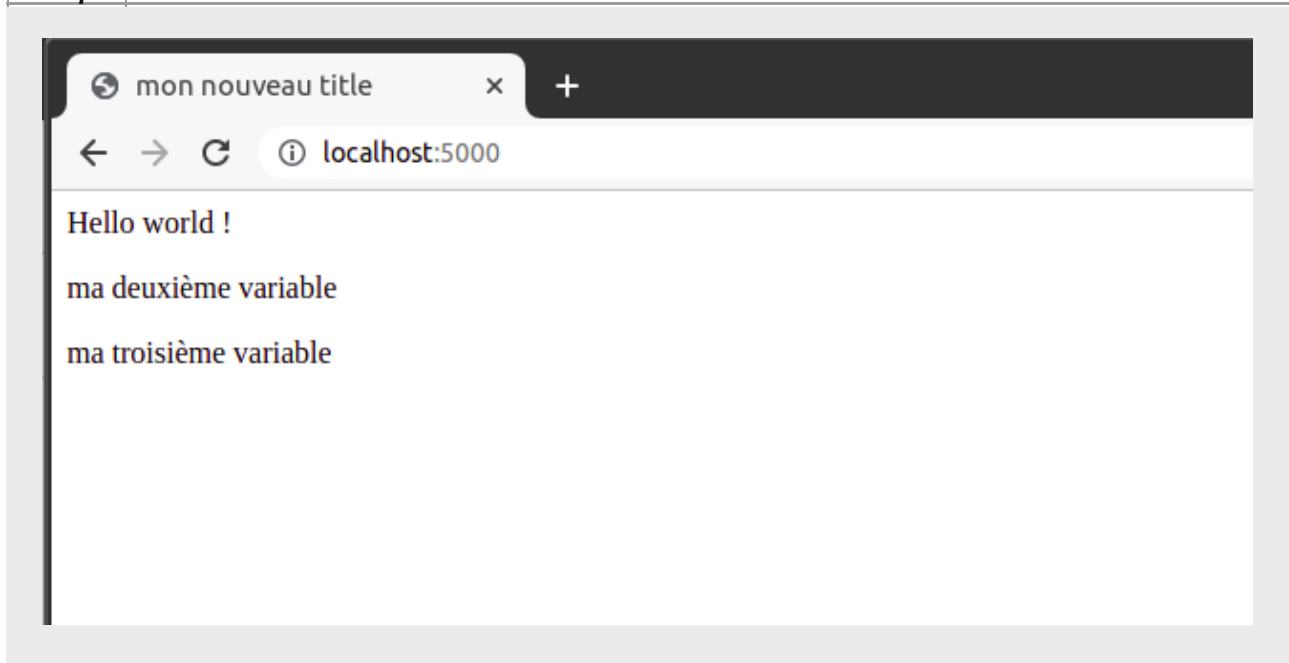
Pour afficher les variables que l'on vient d'ajouter, il faudra encore les ajouter dans la page HTML `homePage.html` (lignes 11 et 12) :

Méthode

```
flask > templates > myTemplate.html > html
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <title>{{myTitle}}</title>
8 </head>
9 <body>
10     Hello world !
11     <p> {{myVariable2}} </p>
12     <p> {{myVariable3}} </p>
13 </body>
14 </html>
```

Fondamental

En exécutant le code de notre page web sur notre *localhost*, nous voyons le contenu des variables «*myVariable2*» et «*myVariable3*» qui s'affichent :

Exemple**Exercice : Quiz**

[solution n°1 p.21]

Question 1

Jinja2 est un moteur de *template* que l'on peut utiliser dans le langage PHP également.

- ☐ Vrai
- ☐ Faux

Question 2

On peut déclarer une nouvelle variable directement dans un *template*.

- ☐ Vrai
- ☐ Faux

Question 3

Pour utiliser une variable dans un *template*, on utilise le tag `{{variable}}`.

- ☐ Vrai
- ☐ Faux

Question 4

On peut créer un *template* et l'ajouter directement dans le dossier du projet.

- ☐ Vrai
- ☐ Faux

Question 5

```
1 @skills_app.route("/")
2 def homepage():
3     return render_template('homePage.html', myTitle='Page d\'accueil')

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <title>{{myTitle}}</title>
8 </head>
9 <body>
10     Hello world !
11 </body>
12 </html>
```

Selon le code ci-dessus, le titre de notre page sera :

- ☐ Page d'accueil
- ☐ myTitle
- ☐ null

III. Apprenez à utiliser l'héritage, les blocs et les boucles

A. L'héritage en Jinja

Fondamental

Comme nous l'avons déjà vu, l'objectif principal des *templates* est de pouvoir générer plusieurs pages web de façon dynamique en se basant sur un ou plusieurs *templates*. Pour cela, nous allons maintenant apprendre à utiliser l'héritage et les *blocs* dans le but d'utiliser un *template* parent qui contiendra ce qui ne sera pas amené à changer dans toutes les autres pages (un squelette).

Dans allons donc dans un premier temps créer une deuxième page HTML, qu'on appellera *about.html*.

Méthode

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <title> About us page </title>
8   </head>
9   <body>
10     Hello world from about us page !
11   </body>
12 </html>
```


Fondamental

Et pour pouvoir l'afficher sur notre navigateur, nous allons améliorer la partie *back-end* en ajoutant la route « */about* » dans le script de notre serveur, afin d'envoyer le *template* *about.html* :

Exemple

```
1 server.py :
2
3 @skills_app.route("/about")
4 def aboutpage():
5     return render_template('about.html')
```

Fondamental

Ensuite, nous ajoutons dans le dossier */templates* une troisième page *base.html* qui sera le *template* sur lequel nous allons nous baser pour générer le contenu des autres pages *html*.

Méthode

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <title>Base template page</title>
8 </head>
9 <body>
10     {% block body %}
11     {% endblock %}
12 </body>
13 </html>
```

Fondamental

La partie *body* du code source de notre page HTML est désormais devenue dynamique grâce aux lignes 10 et 11. *{% block body %}* permet de définir le *bloc body* et *{% end block %}* correspond à sa fin.

Dès à présent nous avons la possibilité d'enlever les parties communes et celles qui se répètent dans chaque page HTML. Dans ce cas, il n'y a que le corps, *body*, qui nous intéresse car la partie *<head> </head>* ne change pas dans les pages *about.html* et *homepage.html*

about.html :

Méthode

Dans :

```
1 templates/about.html :
2
3 {% extends baseTemplate.html' %}
4 {% block body %}
5
6 <p>Hello world from <strong>About us page</strong></p>
7
8 {% endblock %}
```

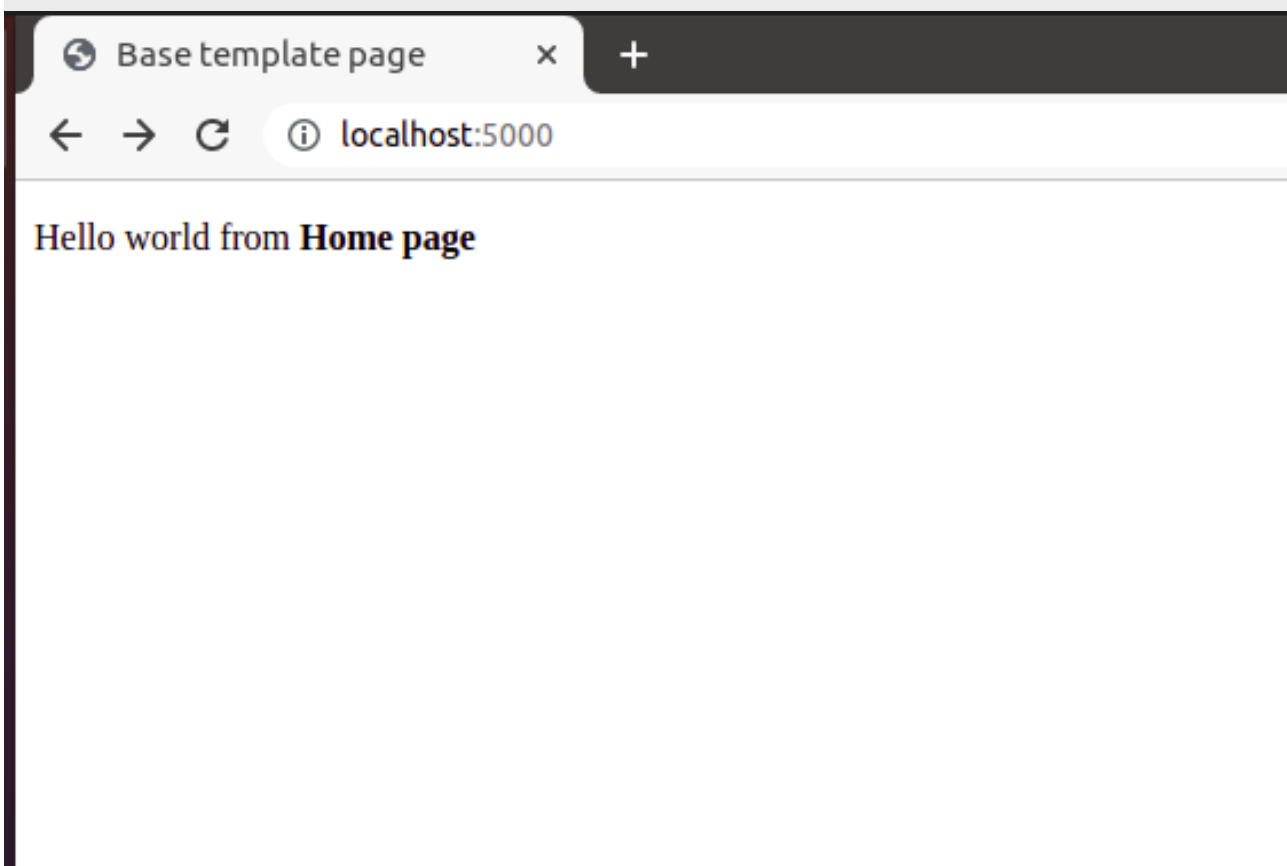
homepage.html :

```
1 {% extends 'base.html' %}
2 {% block body %}
3
4 <p>Hello world from <strong>Home page</strong></p>
5
6 {% endblock %}
```

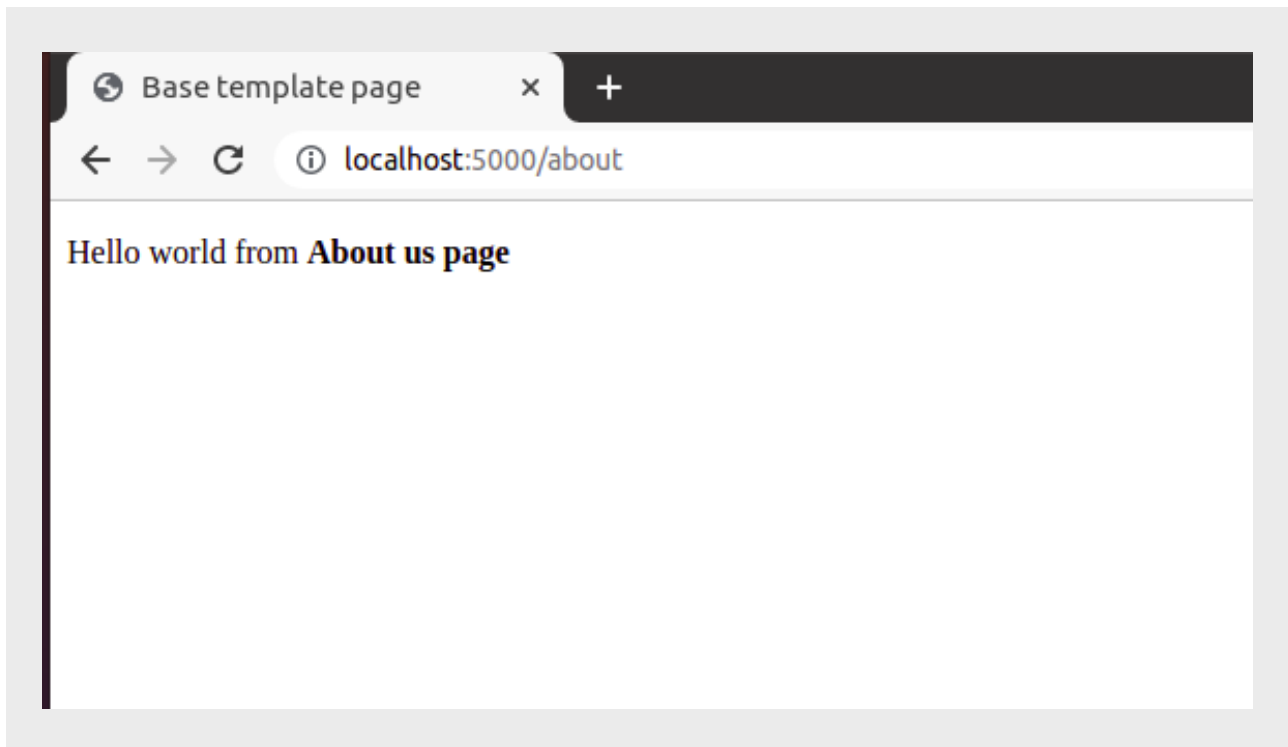
Fondamental

Lorsque l'on affiche les résultats sur le navigateur, on voit bien que les deux pages utilisent le même *template* mais qu'elles affichent 2 résultats différents.

Exemple



About.html page



B. L'ajout du CSS

Fondamental

Afin d'utiliser un fichier CSS, nous devons créer un dossier `/static` dans notre projet (à côté du dossier `/templates` que nous avons déjà créé au tout début du cours). Flask va donc aller chercher automatiquement ce dossier afin de lier le fichier CSS à notre *template*. L'utilisation de la méthode `url_for()` dans l'attribut `href` est indispensable. Elle fait appel aux fichiers qui se trouvent dans le dossier `/static` et elle prend en paramètre le nom du dossier contenant les fichiers CSS, et le nom du fichier CSS que l'on souhaite utiliser (le deuxième paramètre est optionnel).

```
1
2 <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
3
```

C. Les boucles en Jinja

Fondamental

Tout comme pour les autres langages de programmation, les boucles permettent tout simplement de répéter plusieurs fois une série d'instructions. Jinja2 nous donne la possibilité d'utiliser la boucle `for`, sa structure est définie de la façon suivante :

```
1 {% for ... in ... %} ... {% endfor %}
```

Admettons que nous souhaitons récupérer une liste de données et la parcourir afin d'afficher les informations dont l'utilisateur a besoin, l'utilisation de la boucle `for` est donc fortement conseillée afin de boucler sur toutes les valeurs de notre objet.

Dans l'exemple ci-dessous, nous allons essayer d'envoyer au *template* une liste de villes, `countries_list`, et utiliser la boucle `for` pour afficher les résultats.

Méthode

```
1 from flask import Flask, render_template
2
3 skills_app = Flask(__name__)
4
5 countries_list = [
6     {"city": "Paris", "country": "France"},
7     {"city": "Londres", "country": "Angleterre"},
8     {"city": "Munich", "country": "Allemagne"},
9     {"city": "Madrid", "country": "Espagne"}
10 ]
11
12 @skills_app.route("/")
13 def homepage():
14     return render_template('template.html', countries=countries_list)
15
16 if __name__ == "__main__":
17     skills_app.run(debug = True)
```

Fondamental

Nous créons un nouveau *template*: `template.html`, dans lequel nous ajoutons les éléments `` `` et `` `` qui représentent une liste.

Notre boucle `for` va entourer la portion de code qui se répétera (il s'agit ici de l'élément `` de notre liste).

Méthode

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
7     <title>Base template page</title>
8   </head>
9   <body>
10    <ul id="list">
11      {% for item in countries %}
12        <li>Ville: {{ item.city }} || Pays: {{ item.country }}</li>
13      {% endfor %}
14    </ul>
15  </body>
16 </html>
```

Maintenant, il suffit d'actualiser notre page pour obtenir le résultat.

Exemple

← → ↻ ⓘ localhost:5000

- Ville: Paris || Pays: France
- Ville: Londres || Pays: Angleterre
- Ville: Munich || Pays: Allemagne
- Ville: Madrid || Pays: Espagne

D. Les conditions en Jinja

Fondamental

Afin de contrôler l'affichage de certains éléments et vérifier l'existence d'une variable, Jinja offre une structure permettant d'ajouter des conditions `if`.

Si on prend l'exemple précédent, nous pouvons tout d'abord améliorer la qualité de notre code en y ajoutant une condition qui vérifie l'existence du tableau envoyé avant d'itérer sur ses éléments. Pour quel objectif faisons-nous cela ? Tout simplement pour éviter le cas d'erreur suivant :

Exemple



NameError

NameError: name 'countries_list' is not defined

Traceback (most recent call last)

```
File "/home/user1/.local/lib/python3.8/site-packages/flask/app.py", line 2088, in __call__
    return self.wsgi_app(environ, start_response)
File "/home/user1/.local/lib/python3.8/site-packages/flask/app.py", line 2073, in wsgi_app
    response = self.handle_exception(e)
File "/home/user1/.local/lib/python3.8/site-packages/flask/app.py", line 2070, in wsgi_app
    response = self.full_dispatch_request()
File "/home/user1/.local/lib/python3.8/site-packages/flask/app.py", line 1515, in full_dispatch_request
    rv = self.handle_user_exception(e)
File "/home/user1/.local/lib/python3.8/site-packages/flask/app.py", line 1513, in full_dispatch_request
    rv = self.dispatch_request()
File "/home/user1/.local/lib/python3.8/site-packages/flask/app.py", line 1499, in dispatch_request
    return self.ensure_sync(self.view_functions[rule.endpoint])(**req.view_args)
File "/home/user1/dev/proj/courses/flask/flaskApp.py", line 9, in homepage
    return render_template('template.html', countries=countries_list)
```

NameError: name 'countries_list' is not defined

The debugger caught an exception in your WSGI application. You can now look at the traceback which led to the error.

To switch between the interactive traceback and the plaintext one, you can click on the "Traceback" headline. From the text traceback you can also create a paste of it. For code execution mouse-over the frame you want to debug and click on the console icon on the right side.

You can execute arbitrary Python code in the stack frames and there are some extra helpers available for introspection:

- `dump()` shows all variables in the frame
- `dump(obj)` dumps all that's known about the object

Fondamental

Ici, j'ai essayé de retirer la liste `countries_list`, par conséquent le code essaie de boucler sur une variable qui n'existe pas ! Cela peut arriver par exemple lorsque nous avons un problème survenu lors de la récupération des données de la base de données ou d'une API.

Le `body` de notre page `template.html` va donc changer :

Méthode

```

1 <body>
2     {% if countries %}
3     <ul id="list">
4         {% for item in countries %}
5             <li>Ville: {{ item.city }} || Pays: {{ item.country }}</li>
6         </ul>
7     {% endfor %}
8
9     {% else %}
10        Il n'y a rien à afficher
11    {% endif %}
12 </body>

```

Avant de terminer cette partie, je souhaite que l'on ajoute une autre condition afin d'afficher seulement la ville de Paris :

```

1 <!DOCTYPE html>
2 <html lang="en">
3     <head>
4         <meta charset="UTF-8">
5         <meta http-equiv="X-UA-Compatible" content="IE=edge">
6         <meta name="viewport" content="width=device-width, initial-scale=1.0">
7         <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
8         <title>Base template page</title>
9     </head>
10    <body>
11        {% if countries %}
12            <ul id="list">
13                {% for item in countries %}
14                    {% if item.city == "Paris" %}
15                        <li>Ville: {{ item.city }} || Pays: {{ item.country }}</li>
16                    {% endif %}
17                </ul>
18            {% endfor %}
19        {% else %}
20            Il n'y a rien à afficher
21        {% endif %}
22    </body>
23 </html>

```

Attention

Notez bien qu'en Jinja, pour :

- Afficher les variable il faut utiliser :
`{{ Variables }}`
- Pour les déclarations, les boucles, les conditions, etc., il faut utiliser :
`{% Statements %}`

- Enfin pour les commentaires, il faut utiliser :

```
{# Comments #}
```

Exercice : Quiz

[solution n°2 p.22]

Question 1

Pour inclure une variable dans un *template*, il faut utiliser le tag :

- ☐ `{{% ma_variable %}}`
- ☐ `{# ma_variable #}`
- ☐ `{{ ma_variable }}`
- ☐ `[ma_variable]`

Question 2

`render_template` est un objet qui permet de mettre à jour un *template*.

- ☐ Vrai
- ☐ Faux

Question 3

Pour utiliser l'héritage et inclure un *template* parent dans une page HTML, il faut utiliser le tag :

- ☐ `{% extends base.html %}`
- ☐ `{{ extends base.html }}`
- ☐ `{% include base.html %}`
- ☐ `{% import base.html %}`

Question 4

La syntaxe de la boucle `for` est définie :

```
1 {% for ... of ... %} ... {% endfor %}
```

- ☐ Vrai
- ☐ Faux

Question 5

```
1 from flask import Flask, render_template
2
3 skills_app = Flask(__name__)
4
5 users_list = [
6     {"id_user": 1, "exists": True},
7     {"id_user": 2, "exists": False},
8     {"id_user": 3, "exists": True},
9 ]
10
11 @skills_app.route("/")
```



```
12 def homepage():
13     return render_template('template.html', users = users_list)
14
15 if __name__ == "__main__":
16     skills_app.run(debug = True)
17
1 <!DOCTYPE html>
2 <html lang="en">
3     <head>
4         <meta charset="UTF-8">
5         <meta http-equiv="X-UA-Compatible" content="IE=edge">
6         <meta name="viewport" content="width=device-width, initial-scale=1.0">
7         <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
8         <title>Base template page</title>
9     </head>
10    <body>
11        {% if users %}
12        <ul id="list">
13            {% for item in users %}
14                {% if item.exists %}
15                    <li>{{ item.id_user }}</li>
16                {% endif %}
17            </ul>
18            {% endfor %}
19        {% endif %}
20    </body>
21 </html>
```

Le code ci-dessus va afficher les id des utilisateurs :

- ☐ 1 et 3.
- ☐ 1 et 2.
- ☐ 2 et 3.
- ☐ 1, 2 et 3.

V. Essentiel

Jinja2 est un moteur de *template* qui permet de construire un document HTML à partir des éléments suivants :

- Une application Flask qui appelle la fonction `render_template`,
- Un *template* HTML.

Jinja2 permet également d'utiliser des tags particuliers afin :

- D'utiliser des variables et inclure leurs contenus dans un *template*,
- D'utiliser les structures de contrôles (Boucles et Conditions).

VI. Auto-évaluation

A. Exercice

Fabien est un apprenti en développement web, à l'issue de sa formation en entreprise il souhaite commencer la création d'un site web. Pour cela, il effectue des recherches sur les différentes technologies à utiliser. Fabien a finalement pris la décision d'utiliser le *framework* Flask et le *template* Jinja2, c'est pourquoi il voudrait savoir :

Question 1

[solution n°3 p.24]

Quelles sont les étapes nécessaires à suivre afin de générer un *template* Jinja2 en utilisant le *framework* Flask ?

Question 2

[solution n°4 p.24]

Peut-on utiliser les *templates* dans d'autres langages de programmation ?

B. Test

Exercice 1 : Quiz

[solution n°5 p.24]

Question 1

La fonction `render_template` est appelée avec un seul paramètre (le nom de la *template*).

- ☐ Vrai
- ☐ Faux

Question 2

Les instructions suivantes permettent de définir le bloc « body » à l'intérieur du *template* « base.html ».

```
1 {% block body %}
2 {% endblock %}
```

- ☐ Vrai
- ☐ Faux

Question 3

Les instructions suivantes permettent de définir le bloc « body » à l'intérieur du *template* « base.html ».

```
1 {% extend "template.html" %}
```

- ☐ Vrai
- ☐ Faux

Question 4

Pour inclure une condition dans un *template*, il faut utiliser le tag :

- ☐ `{{% if %}} {{% else %}}`
- ☐ `{# if #} {# else #}`
- ☐ `{{ if }} {{ else }}`
- ☐ `[if] [else]`

Question 5

La syntaxe de la boucle `for` est définie :

```
1 {% for ... in ... %} ... {% endfor %}
```

- ☐ Vrai
- ☐ Faux

Solutions des exercices

Exercice p. 7 Solution n°1**Question 1**

Jinja2 est un moteur de *template* que l'on peut utiliser dans le langage PHP également.

☐ Vrai

☒ Faux


 Jinja2 est conçu pour être utilisé seulement en Flask (Python).

Question 2

On peut déclarer une nouvelle variable directement dans un *template*.

☐ Vrai

☒ Faux


 Il n'est pas possible de déclarer une variable directement dans un *template*, cependant nous avons la possibilité de passer une variable via la fonction *render_template*. Les variables doivent donc être déclarées dans la partie *back-end*.

Question 3

Pour utiliser une variable dans un *template*, on utilise le tag `{{variable}}`.

☒ Vrai

☐ Faux


 Le tag `{{variable}}` permet d'utiliser les variables passées via la fonction *render_template*.

Question 4

On peut créer un *template* et l'ajouter directement dans le dossier du projet.

☐ Vrai

☒ Faux

 Pour créer un *template*, il faut tout d'abord commencer par créer un dossier */templates*. Ensuite, à l'intérieur de ce dossier, on peut ajouter le *template* que l'on souhaite créer.

Question 5


```
1 @skills_app.route("/")
2 def homepage():
3     return render_template('homePage.html', myTitle='Page d\'accueil')

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <title>{{myTitle}}</title>
8 </head>
```

```
9 <body>
10   Hello world !
11 </body>
12 </html>
```

Selon le code ci-dessus, le titre de notre page sera :

- ☒ Page d'accueil
- ☐ myTitle
- ☐ null


 Dans la fonction `render_template`, on passe la variable `myTitle` qui contient une chaîne de caractères (Page d'accueil), le titre de notre page sera « Page d'accueil ».

Exercice p. 16 Solution n°2

Question 1

Pour inclure une variable dans un *template*, il faut utiliser le tag :


- ☐ `{{% ma_variable %}}`
- ☐ `{# ma_variable #}`
- ☒ `{{ ma_variable }}`
- ☐ `[ma_variable]`

 Pour afficher le contenu d'une variable, il faut utiliser le tag `{{ ma_variable }}`.

Question 2

`render_template` est un objet qui permet de mettre à jour un *template*.


- ☐ Vrai
- ☒ Faux

 `render_template` une méthode qui permet de retourner un *template* afin de l'afficher, elle est appelée avec deux paramètres : le nom du *template* et la liste des variables à utiliser par le *template*.

Question 3

Pour utiliser l'héritage et inclure un *template* parent dans une page HTML, il faut utiliser le tag :

- ☒ `{% extends base.html %}`
- ☐ `{{ extends base.html }}`
- ☐ `{% include base.html %}`
- ☐ `{% import base.html %}`

 Pour inclure un *template* parent dans une page HTML, on utilise le tag `{% extends base.html %}`.


Question 4

La syntaxe de la boucle `for` est définie :

```
1 {% for ... of ... %} ... {% endfor %}
```

☐ Vrai

☒ Faux

 La boucle `for` utilise une structure définie par le tag :

```
1 {% for ... in ... %} ... {% endfor %}
```


Question 5

```
1 from flask import Flask, render_template
2
3 skills_app = Flask(__name__)
4
5 users_list = [
6     {"id_user": 1, "exists": True},
7     {"id_user": 2, "exists": False},
8     {"id_user": 3, "exists": True},
9 ]
10
11 @skills_app.route("/")
12 def homepage():
13     return render_template('template.html', users = users_list)
14
15 if __name__ == "__main__":
16     skills_app.run(debug = True)
17
```

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <link rel="stylesheet" href="{% url_for('static', filename='css/style.css') %}">
8     <title>Base template page</title>
9   </head>
10  <body>
11    {% if users %}
12    <ul id="list">
13      {% for item in users %}
14        {% if item.exists %}
15        <li>{{ item.id_user }}</li>
16        {% endif %}
17      </ul>
18    {% endfor %}
19    {% endif %}
20  </body>
21 </html>
```

Le code ci-dessus va afficher les id des utilisateurs :

- ☒ 1 et 3.
- ☐ 1 et 2.
- ☐ 2 et 3.
- ☐ 1, 2 et 3.

 Le code affichera les ids 1 et 3 car dans le *template*, on vérifie la condition :

```
1 {% if item.exists %}
```

Par conséquent, les utilisateurs ayant la propriété *exists* égale à True sont ceux qui possèdent les ids 1 et 3.

p. 18 Solution n°3

Pour générer un *template* Jinja2, il faut tout d'abord commencer par coder la partie *back-end* en y ajoutant une route qui permettra d'envoyer notre *template*.

Il faut faire appel à la méthode *render_template* qui prend en paramètre le nom du *template* à envoyer et la liste des variables à utiliser par le *template*.

Une fois la partie *back-end* faite, il faudra ensuite créer un dossier « /templates » et à l'intérieur de ce dernier, nous allons créer notre *template* « myTemplate.html ».

Il ne reste plus qu'à ajouter le code HTML avec les différents tags à utiliser (conditions, boucles, etc.) dans notre *template*.

p. 18 Solution n°4

Oui effectivement, on trouve le principe des *templates* dans d'autres langages de programmation, par exemple en PHP nous avons la possibilité d'utiliser le *template* Twig.


Pour le *framework* Flask, nous utilisons le *template* Jinja2.

Exercice p. 18 Solution n°5

Question 1

La fonction *render_template* est appelée avec un seul paramètre (le nom de la *template*).

- ☐ Vrai
- ☒ Faux


 *render_template* est appelée avec deux paramètres : le nom du *template* et la liste des variables à utiliser par le *template*.

Question 2

Les instructions suivantes permettent de définir le bloc « body » à l'intérieur du *template* « base.html ».

```
1 {% block body %}
2 {% endblock %}
```

- ☒ Vrai
- ☐ Faux

 La syntaxe du code est correcte.


Question 3

Les instructions suivantes permettent de définir le bloc « body » à l'intérieur du *template* « base.html ».

```
1 {% extend "template.html" %}
```

☐ Vrai

☒ Faux

 Il manque un « s » à la fin du mot clé « *extends* ».

Question 4


Pour inclure une condition dans un *template*, il faut utiliser le tag :

☒ `{{% if %}} {{% else %}}`

☐ `{# if #} {# else #}`

☐ `{{ if }} {{ else }}`

☐ `[if] [else]`

 Pour inclure une condition dans un *template*, il faut utiliser le tag `{{% if %}} {{% else %}}`.


Question 5

La syntaxe de la boucle `for` est définie :

```
1 {% for ... in ... %} ... {% endfor %}
```

☒ Vrai

☐ Faux

 La boucle `for` utilise la structure :

```
1 {% for ... in ... %} ... {% endfor %}
```