

# Cadre théorique de flask

# Table des matières

<b>I. Cadre théorique de Flask</b>	<b>3</b>
A. Python.....	3
B. Flask.....	4
C. Framework Web .....	4
D. Composants de Flask.....	4
E. Notre première application Flask .....	5
<b>II. Exercice : Exercice applicatif : « Appliquer la notion »</b>	<b>6</b>
<b>III. Templates</b>	<b>6</b>
A. Templates en Flask.....	7
B. Système des templates Jinja .....	7
C. Layouts, pages et partiels .....	8
<b>IV. Exercice : Exercice applicatif : « Appliquer la notion de templates »</b>	<b>11</b>
<b>V. Routage</b>	<b>11</b>
A. Routes fixes.....	12
B. Routes Dynamiques .....	12
<b>VI. Exercice : Exercice applicatif : « Appliquer la notion de redirection »</b>	<b>13</b>
<b>VII. Essentiel</b>	<b>14</b>
<b>VIII. Auto-évaluation</b>	<b>14</b>
A. Exercice .....	14
B. Test.....	14
<b>Solutions des exercices</b>	<b>16</b>

# I. Cadre théorique de Flask

**Durée** : 50 minutes

**Environnement de travail** : Un éditeur de texte

**Prérequis** : Notions en Python

## Contexte

Python est un langage de programmation puissant, de plus en plus utilisé par les chercheurs scientifiques, les ingénieurs logiciels, les data scientists et aussi par les débutants en programmation. Flask est un Framework Web qui aide à créer des applications Web complètes à l'aide de Python. Dans ce cours nous allons apprendre les bases de la création d'applications Web avec Flask. À la fin de ce cours nous serons capables de créer nos propres sites avec des interfaces interactives et une navigation personnalisée entre nos pages.

## Objectifs

- Connaître les notions de base de Flask
- Comprendre la notion de framework
- Installer l'environnement logiciel pour le cours

## Contexte

Dans le contexte d'une pratique professionnelle, vous êtes amenés à connaître les notions théoriques derrière tout outil que vous allez utiliser, vous devez également être en mesure de préparer votre environnement de travail : installation des outils à utiliser.

## A. Python

Python est un langage de programmation interprété (pas de phase de compilation comme pour les programmes C / C++) avec des structures de données intégrées de haut niveau assez attrayantes pour le développement rapide d'applications. L'interpréteur Python ainsi que les bibliothèques standards sont disponibles sous forme source ou binaire sur toutes les principales plates-formes (Linux, Windows ou Mac).

Pour exécuter un programme Python, nous devons avoir l'interpréteur Python disponible dans le terminal ou l'invite de commande. La première étape consiste à accéder à la page de téléchargement officielle, à télécharger le programme d'installation de notre système d'exploitation (Mac, Linux, Windows) et à l'exécuter. Si l'installation se passe bien on devrait pouvoir accéder à l'interpréteur depuis la fenêtre du terminal (Powershell pour windows, Xterm pour Linux, etc.).

```
$ python --version
$ Python 3.7.2
```

Veuillez installer la version Python 3.x, Python 2.x est obsolète et n'est plus pris en charge par la fondation Python.

## B. Flask

Flask est un framework Web Python construit avec un petit noyau et une modularité. Il est classé comme un micro-framework car il ne nécessite pas d'outils ou de bibliothèques particuliers. Il n'a pas de couche d'abstraction de base de données, de validation de formulaire ou tout autre composant, et toutes ces fonctionnalités nécessaires sont fournies par des bibliothèques tierces développées par l'équipe derrière Flask ou d'autres développeurs de la communauté.



**Project Links**

- [Donate](#)
- [PyPI Releases](#)
- [Source Code](#)
- [Issue Tracker](#)
- [Website](#)
- [Twitter](#)
- [Chat](#)

**Contents**

- [Welcome to Flask](#)
- [User's Guide](#)
- [API Reference](#)
- [Additional Notes](#)

**Quick search**

**Flask**  
web development,  
one drop at a time

Welcome to Flask's documentation. Get started with [Installation](#) and then get an overview with the [Quickstart](#). There is also a more detailed [Tutorial](#) that shows how to create a small but complete application with Flask. Common patterns are described in the [Patterns for Flask](#) section. The rest of the docs describe each component of Flask in detail, with a full reference in the [API](#) section.

Flask depends on the [Jinja](#) template engine and the [Werkzeug](#) WSGI toolkit. The documentation for these libraries can be found at:

- [Jinja documentation](#)
- [Werkzeug documentation](#)

**User's Guide**

This part of the documentation, which is mostly prose, begins with some background information about Flask, then focuses on step-by-step instructions for web development with Flask.

- [Foreword](#)

## C. Framework Web

Un framework Web est un ensemble de composants et de bibliothèques conçus pour simplifier votre processus de développement Web.

En effet, le but d'un Web Framework est de nous faciliter la vie et d'apporter une aide concernant la configuration répétitive et les modules requis par plusieurs projets. Flask le fait parfaitement.

## D. Composants de Flask

Le micro-framework Flask fait partie des Pallets Projects (Pocoo) et il est basé sur un nombre de ces projets :

- Werkzeug : permet la communication entre les serveurs Web et les applications Python.
- Jinja : Jinja va nous aider à écrire des fichiers HTML qui pourront exécuter des instructions en Python, en effet c'est un moteur de templates rapide, expressif et extensible.
- MarkupSafe : MarkupSafe implémente un objet texte qui saute les caractères afin qu'il puisse être utilisé en toute sécurité en HTML et XML. Les caractères qui ont des significations spéciales sont remplacés afin qu'ils s'affichent comme les caractères réels. Cela atténue les attaques par injection, ce qui signifie que les entrées utilisateurs non fiables peuvent être affichées en toute sécurité sur une page.
- ItsDangerous : c'est une bibliothèque de la sérialisation sécurisée de données destinée au langage de programmation Python. Il est utilisé pour stocker la session d'une application Flask dans un cookie sans permettre aux utilisateurs de falsifier le contenu de la session.

## E. Notre première application Flask

Nous allons maintenant créer notre programme Flask « *Hello world !* ».

Après l'installation de python, nous devons installer Flask. Pour ce faire il suffit d'exécuter dans le terminal la commande suivante :

```
1 Shell( <pip install Flask>
```

Après dans un éditeur de code on écrit notre application « *Hello world !* ».

```
1 Python(< from flask import Flask
2
3 app = Flask(__name__)
4
5 @app.route('/')
6 def hello():
7     return 'Hello World !' >
```

Nous allons expliquer les lignes importantes de ce code :

`app = Flask(__name__)`

C'est la ligne qui crée une nouvelle application : ce dont on a besoin pour exécuter le processus. Elle permet aussi de traiter les différentes tâches indispensables pour notre application Web. 'name' est une variable automatique définie par Python et recommandée pour la création des applications Flask.

`@app.route('/')`

En Flask, cette ligne, ajoutée en dessus d'une fonction, permet de définir dans quelle route la fonction sera invoquée.

`def hello():`

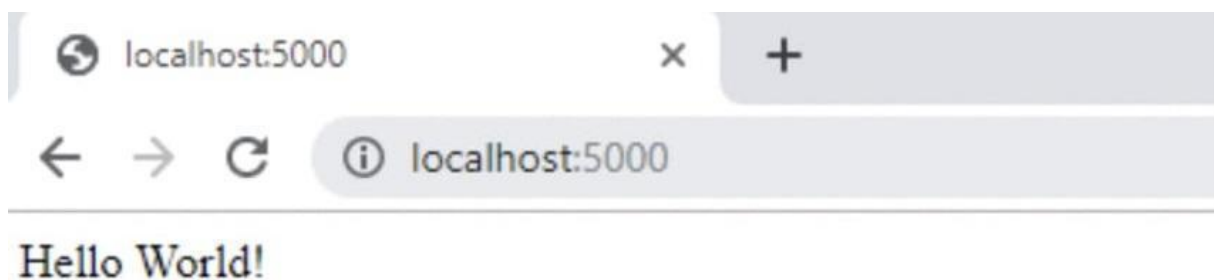
`return 'Hello World !'`

C'est notre fonction qui ne prend aucun paramètre et retourne « *Hello World !* ».

Nous sauvegardons le fichier sous « *hello\_world.py* » et nous démarrons l'application.

Shell( env FLASK\_APP=hello\_world.py flask run.

Pour visiter l'application, il suffit d'aller sur localhost:5000 à l'aide d'un navigateur web.



Nous pouvons modifier les paramètres de l'application. Le code ci-dessous est un exemple de personnalisation de notre application Flask.

```
1 Python(< app = Flask( __name__ )
2
3 @app.route('/')
4 def hello():
5     return 'Hello World !'
6 if __name__ == '__main__':
7
8 app.run(
9     host='0.0.0.0',
```

```
10 debug=True,
11 port=8080
12 ) >
```

Les projets Flask s'exécutent par défaut dans l'URL « 127.0.0.1:5000 », si nous voulons changer de port nous devons définir l'host et le port dans `app.run(...)`. Nous ajoutons également `debug=True` qui va nous permettre d'apporter automatiquement les modifications rien qu'en actualisant la page.

#### Complément

Documentation officielle de Flask<sup>1</sup>

## II. Exercice : Exercice applicatif : « Appliquer la notion »

### Question 1

[solution n°1 p.17]

Qu'est-ce que Flask ?

### Question 2

[solution n°2 p.17]

Pourquoi l'utiliser ?

### Question 3

[solution n°3 p.17]

Quel est le port hôte par défaut de Flask ?

### Question 4

[solution n°4 p.17]

Comment changer l'hôte et le port dans Flask ?

## III. Templates

### Objectifs

- Comprendre Jinja en tant que moteur de templates
- Apprendre à afficher des pages HTML avec Jinja
- Apprendre à communiquer des données de et vers des pages HTML
- Apprendre à construire une application Web à partir de plusieurs pages

#### Contexte

Dans le contexte d'une pratique professionnelle basée sur le développement Web à l'aide de Flask, vous utiliserez la notion de templates pour pouvoir créer une multitude de pages dans un site Web, et définir leur contenu. Vous allez également pouvoir créer du contenu réutilisable entre plusieurs pages, cette partie de cours vous sera aussi utile dans ce sens.

<sup>1</sup> <https://flask.palletsprojects.com/en/2.0.x/>

## A. Templates en Flask

Avant de commencer l'utilisation des templates, nous allons coder une simple application qui rend une page sans template. Pour ce faire nous allons exécuter le code suivant :

```
1 Python(<from flask import Flask
2 app = Flask(__name__)
3
4 @app.route('/')
5 def hello():
6     return '''
7 <html>
8     <head>
9         <title>Notre application Flask</title>
10    </head>
11    <body>
12        <h1>Bonjour</h1>
13        <p>Notre application Flask - sans template</p>
14    </body>
15 </html>'''
```

Ce que nous avons fait est de retourner une chaîne de caractère sous format d'une page HTML avec toutes les sections habituelles : head, body, etc.



Mais lorsque l'on travaille sur un projet qui contient plusieurs pages, il serait plus pratique d'utiliser un système de templates, pour avoir un projet bien organisé et maintenable.

## B. Système des templates Jinja

Un système de templates nous permet de travailler avec des templates Web pour générer automatiquement des pages Web personnalisées, telles que les résultats d'une recherche. Cela réutilise les éléments de page Web statiques tout en définissant des éléments dynamiques.

Jinja est une bibliothèque Python utilisée par des frameworks Web populaires comme Flask et Django pour servir des pages HTML de manière sécurisée et efficace. L'utilisation d'un moteur de templates est une pratique courante dans le développement Web, quel que soit le langage ou le framework utilisé pour coder l'application Web.

Pour tester le moteur Jinja nous allons exécuter dans une console Python :

```
1 Shell(< from jinja2 import Template
2 t = Template("Test de {{ application }}")
3 t.render(application="Jinja")>
```

Jinja nous permet de retourner des templates, qui sont principalement des fichiers HTML en utilisant la méthode : `render_template`. Voici un exemple de code avec cette méthode :

```
1 Python(<from flask import render_template
2
3 @app.route('/')
4 def monTemplate():
5     render_template('nom_de_mon_template.html') >
```

Souvent dans nos applications, nous ne voulons pas retourner des pages HTML statiques. En Flask, il est possible de passer des données aux templates.

Dans cet exemple de code, nous allons définir une variable `ami` qu'on va passer au template :

```
1 Python(<from flask import render_template
2 @app.route('/')
3 def bonjour():
4     ami = 'Jean'
5     render_template('bonjour.html' , 'ami' : ami >
```

Maintenant, nous allons passer la donnée `ami` à notre template :

```
1 HTML(< <h1> Bonjour {{ ami }} !</h1>>
```

## C. Layouts, pages et partiels

Nous pouvons classer les fragments d'HTML comme appartenant à l'un des 3 types suivants :

- **Layout** : la majorité des applications Web ont une sorte de structure « *squelette* » partagée entre les pages, telle qu'un contenu standard ou une barre latérale. Il est plus simple d'appliquer une structure générale que de répéter la même chose sur plusieurs pages. Une application Web cohérente a généralement très peu de mises en page uniques.
- **Page** : une page est une section assemblée de manière unique et qui peut être répliquée avec différentes données.
- **Partiels** : les partiels sont des extraits autonomes qui peuvent être partagés par des pages si nécessaire. Comme les éléments de navigation, les widgets ou à tout autre élément conçu pour être partagé entre les parties de notre site.

Nous allons créer une petite application Flask pour montrer comment utiliser Jinja pour créer des sites Web en utilisant les trois concepts que nous venons d'apprendre. Nous allons commencer par créer une application qui utilise chacun des trois types de modèles : `layout.html`, `home.html` et `navigation.html`.

Ces modèles seront dans le dossier `/templates`, la structure de l'application ressemble à ceci :

```
/flask_exemple
├── /flask_exemple
│   ├── __init__.py
│   ├── routes.py
│   ├── /static
│   └── /templates
│       ├── home.html
│       ├── layout.html
│       └── navigation.html
└── wsgi.py
```



Les templates nous seront utiles au sein d'une application Flask, nous allons tout d'abord créer une application Flask dans `init.py`. Après nous créons `routes.py` pour définir la page d'accueil « *home* » :

```
1 Python(<from flask import current_app as app
2 from flask import render_template
3
4 @app.route('/')
5 def home():
6     return render_template( 'home.html' ,
7                             title="Exemple de templates avec Jinja",
8                             description="Mon application avec Flask et Jinja.") >
```

Tout ce que nous faisons ici, c'est configurer un itinéraire appelé `home()`, qui sert le modèle `home.html` à chaque fois qu'un utilisateur visite notre application Flask. En plus de spécifier quel modèle de page servir, nous passons également deux arguments de mot-clé à `render_template()`: `title` et `description`.

Maintenant nous allons écrire notre page template `home.html`

```
1 HTML(<{% extends 'layout.html' %}
2
3 {% block contenu %}
4     <div class="container">
5         <h1>{{title}}</h1>
6         <p>{{description}}</p>
7     </div>
8 {% endblock %}>
```

Nous avons mentionné que les « *layouts* » contiennent les structures répétitives partagées entre les pages. Ici `{% extends 'layout.html' %}` fait référence à `layout.html`. Pour mieux saisir le concept de « *layout* », regardons un exemple de fichier `layout.html` :

```
1 HTML(< <html>
2
3     <head>
4         <title>{{title}}</title>
5         <meta charset="utf-8">
6         <meta name="description" content={{description}}>
7         <link rel="shortcut icon" href="/favicon.ico">
8     </head>
9
10    <body>
11        {% include 'navigation.html' %}
12        {% block contenu %}{% endblock %}
13    </body>
14
15 </html>>
```

Lorsque nous demandons une page template qui « *étend* » (extends) une autre, la page demandée va contenir tout le contenu HTML de la page qu'elle étend (page mère). Dans notre cas, quand `home.html` étend `layout.html`, nous devrons savoir où exactement dans `layout.html` sera montée. Jinja gère cela en faisant correspondre des espaces réservés dans nos layouts, nommés « *blocs* ».

`layout.html` a un seul bloc appelé « contenu » qui apparaît à la fois dans notre layout de page et dans la mise en page qu'il étend - cela permet à Jinja de savoir que le contenu de `home.html` doit être chargé dans ce bloc - ce qui entraîne effectivement ce qui suit à l'exécution :

```
1 HTML(< <html>
2
3     <head>
4         <title>Exemple de templates avec Jinja</title>
5         <meta charset="utf-8">
6         <meta name="description" content = 'Mon application avec Flask et Jinja.' >
```

```

7     <link rel="shortcut icon" href="/favicon.ico">
8 </head>
9
10 <body>
11     {% include 'navigation.html' %}
12     <div class="container">
13         <h1>Exemple de templates avec Jinja</h1>
14         <p>Mon application avec Flask et Jinja.</p>
15     </div>
16 </body>
17
18 </html>>

```

Ce qui précède montre ce qu'est une page home.html rendue après avoir été chargée dans layout.html. Notre page ressemble déjà beaucoup plus à du HTML.

Cela nous laisse avec une dernière déclaration de Jinja : `{% include 'navigation.html' %}`. include dit à Jinja de charger un template séparé nommé navigation.html ici. Ces templates autonomes qui sont déposés dans des templates plus grands sont appelés partiels. Notre navigation.html est un simple en-tête de navigation qui ressemble à ceci :

```

1 HTML(<header>
2     <nav>
3         <a href="https://exemple1.com"> Lien 1 </a>
4         <a href="https://exemple2.com"> Lien 2 </a>
5         <a href="https://exemple3.com"> Lien 3 </a>
6     </nav>
7 </header>

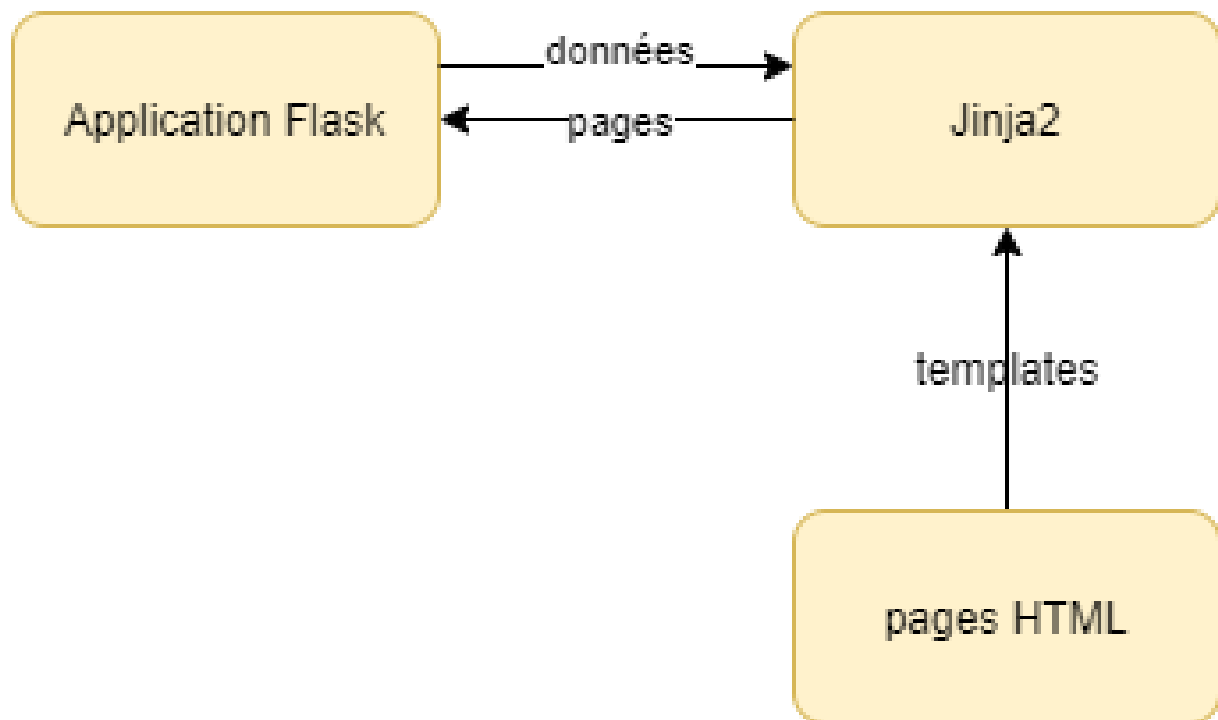
```

Nous pouvons maintenant voir à quoi ressemble notre page d'accueil rendue dans son intégralité :

```

1 HTML(< <html>
2     <head>
3         <title>Exemple de templates avec Jinja</title>
4         <meta charset="utf-8">
5         <meta name="description" content="'Mon application avec Flask et Jinja.'>
6         <link rel="shortcut icon" href="/favicon.ico">
7     </head>
8     <body>
9         <header>
10             <nav>
11                 <a href="https://exemple1.com"> Lien 1 </a>
12                 <a href="https://exemple2.com"> Lien 2 </a>
13                 <a href="https://exemple3.com"> Lien 3 </a>
14             </nav>
15         </header>
16         <div class="container">
17             <h1>Exemple de templates avec Jinja</h1>
18             <p>Mon application avec Flask et Jinja.</p>
19         </div>
20     </body>
21 </html> >

```

**Complément**

Documentation officielle de Jinja<sup>1</sup>

#### IV. Exercice : Exercice applicatif : « Appliquer la notion de templates »

**Question**

[solution n°5 p.17]

On veut créer une page où l'on affiche un élément HTML préalablement élaboré enregistré dans le fichier navigation.html. La page contient le texte : bonjour je m'appelle <nom>, nom va être défini au niveau du code python.

#### V. Routage

**Objectifs**

- Apprendre à naviguer entre les pages d'une application Web Flask
- Apprendre à communiquer des données dans des routes
- Apprendre à utiliser la méthode HTTP POST dans un contexte Flask

**Contexte**

Dans le contexte d'une pratique professionnelle basée sur le développement Web en Flask, vous utiliserez la notion du routage pour définir la navigation entre vos pages et pour communiquer des données entre ces dernières.

<sup>1</sup> <https://jinja.palletsprojects.com/en/3.0.x/>

Le routage est utilisé pour mapper un URL particulier avec la fonction qui lui est affectée pour la réalisation d'une tâche. Elle permet d'accéder à une page particulière.

Dans notre première application « *Hello World* ! », l'URL ('/') est associée à la méthode hello qui renvoie une chaîne particulière affichée sur notre page Web.

Autrement dit, si nous visitons une URL particulière mappée à une méthode particulière, nous aurons la valeur de retour de cette méthode affichée sur l'écran du navigateur.

Nous allons maintenant en apprendre davantage sur les routes.

## A. Routes fixes

Si nous voulons avoir des routes permanentes pour nos pages, nous n'avons qu'à utiliser les routes fixes. La création des URLs fixes est simple :

```
1 Python(<@app.route('/jean">
2 def jean():
3     return 'Bonjour Jean !'
4
5 @app.route('/louise/")
6 def hello():
7     return 'Bonjour Louise !' >
```

Nous pouvons utiliser les deux syntaxes : /ma page ou /ma\_page/. L'utilisation de la première syntaxe renvoie une erreur 404, si nous essayons l'URL /ma\_page/. Tandis que la deuxième syntaxe redirige vers /ma\_page/ en cas de saisie de l'URL /ma\_page.

## B. Routes Dynamiques

### Avec un seul paramètre

Utiliser les routes dynamiques consiste à récupérer des données dynamiques à partir de l'URL, et de les utiliser.

En Flask, nous pouvons utiliser les convertisseurs suivants pour convertir la donnée dynamique récupérée de l'URL.

- string : chaîne des caractères,
- int : un nombre entier,
- float : un nombre décimal,
- path : c'est une chaîne des caractères qui accepte les slashes dans l'URL.

Nous pouvons récupérer les données à partir de l'URL sans avoir recours à l'utilisation de ces paramètres, toutefois, il est recommandé de les utiliser.

Voici deux exemples :

```
1 Python(<@app.route('bonjour/<nom_ami>/")
2 def bonjour(nom_ami):
3     return 'Bonjour' + str(nom_ami) >
```

Dans cet exemple nous avons passé la donnée nom\_ami à la méthode sans utiliser les convertisseurs. Maintenant nous allons utiliser un convertisseur qui va convertir la donnée en string (chaîne des caractères) puis la passer à la méthode.

```
1 Python(<@app.route('bonjour/<str:nom_ami>/")
2 def bonjour(nom_ami):
3     return 'Bonjour' + nom_ami>
```

### Avec plusieurs paramètres

Nous pouvons également passer plusieurs paramètres dans une route en Flask, de la même façon qu'avec un seul paramètre. L'exemple suivant illustre la récupération des données `mon_nom` et `nom_ami`.

```
1 Python(<@app.route('bonjour/<mon_nom>/<nom_ami>/'>)
2 def bonjour(mon_nom,nom_ami):
3     return 'Bonjour' + nom_ami + 'c'est' + mon_nom
```

### Requêtes POST avec des routes

Flask nous offre la possibilité d'utiliser les requêtes POST du protocole HTTP.

Nous allons essayer dans cet exemple de récupérer la donnée `nom` que l'utilisateur de notre application va saisir. D'abord, nous allons créer notre template HTML comme suit :

```
1 HTML(<<html>
2     <body>
3         <form action = "http://localhost:5000/nom" method = "post">
4             <p>Entrer Votre nom:</p>
5             <p><input type = "text" name = "nom" /></p>
6             <p><input type = "submit" value = "submit" /></p>
7         </form>
8     </body>
9 </html>>
```

Le code suivant supporte le type de requêtes POST :

```
1 Python(<from flask import Flask
2 from flask import render_template
3 from flask import request
4
5 from flask import Flask, redirect, url_for, request
6 app = Flask(__name__)
7
8 @app.route('/accueil/<nom>')
9 def accueil(nom):
10     return 'Bonjour' + nom
11
12 @app.route('/nom',methods = ['POST'])
13 def recuperer_nom():
14     utilisateur = request.form['nom']
15     return redirect(url_for('accueil',nom = utilisateur))
16 if __name__ == '__main__':
17     app.run(debug = True) >
```

Ce bout de code que nous venons d'écrire se compose de deux méthodes : `accueil` et `recuperer_nom`. La méthode `recuperer_nom` nous permet de récupérer la donnée `nom` saisie par l'utilisateur avec la méthode POST puis redirige la navigation vers « `/accueil` » qui appelle la méthode `accueil`. Cette dernière affiche un message d'accueil avec le nom reçu en paramètre.

## VI. Exercice : Exercice applicatif : « Appliquer la notion de redirection »

### Question

[solution n°6 p.18]

On veut créer et utiliser la notion de route multiple pour récupérer deux entiers à partir du lien « `addition/nombre_1/nombre_2` », et on veut afficher leur somme. Comment faire ?

## VII. Essentiel

Nous avons appris dans ce cours plusieurs notions relatives au Framework Flask. Nous pouvons résumer ces acquis comme suit :

- Flask est micro-Framework Web qui nous aide en développement de nos applications Web et qui accepte l'ajout de plusieurs librairies tierces.
- La gestion des interfaces Flask est simplifiée avec l'utilisation de Jinja.
- Nous pouvons appeler des fichiers HTML et CSS avec l'outil Jinja, ce qui nous facilite la création des pages de notre application.
- Grâce au routage offert par Flask, nous pouvons créer et afficher des pages dynamiques et statiques.

## VIII. Auto-évaluation

### A. Exercice

#### Question

[solution n°7 p.18]

On veut créer une application de calcul Web avec Flask avec plusieurs pages : une page d'addition, une page de multiplication, une page de soustraction et une page de division. Les nombres sur lesquels on va effectuer nos opérations sont passés en liens (routes). Et les pages affichent le résultat des opérations.

### B. Test

#### Exercice 1 : Quiz

[solution n°8 p.18]

#### Question 1

Comment peut-on activer le débogage dans Flask ?

- ☐ Passer la propriété debug avec la valeur True en paramètre dans la méthode run.
- ☐ On ne peut pas activer le débogage
- ☐ Dans app.route

#### Question 2

À quoi l'activation de débogage dans Flask peut être utile ?

- ☐ L'activation de débogage augmente la sécurité en imposant un redémarrage manuel à chaque modification du code
- ☐ Le serveur se rechargera lorsque le code change et vous n'avez pas besoin de redémarrer manuellement après chaque modification du code
- ☐ L'activation de débogage active automatiquement les tests

#### Question 3

Qu'est-ce que Jinja ?

- ☐ Il crée des éléments logiciels pour des fonctions
- ☐ C'est un moteur de template
- ☐ Il mappe les routes

#### Question 4

Quelle est la syntaxe correcte pour appeler la variable « *var* » dans du code HTML ?

- ☐ #var
- ☐ <<var>>.
- ☐ {{var}}

Question 5

Quelle est la méthode qui permet d'afficher un contenu à partir d'une page HTML ?

- ☐ return\_page()
- ☐ render\_template()
- ☐ show\_content()

Question 6

Comment on peut charger un contenu HTML dans un autre contenu HTML ?

- ☐ Avec include
- ☐ Avec extend
- ☐ En concaténant les deux contenus

Question 7

Comment on peut charger un contenu HTML qui étend un autre contenu HTML ?

- ☐ Avec include
- ☐ Avec extend
- ☐ En concaténant les deux contenus

Question 8

Comment on définit les routes en Flask ?

- ☐ Avec app.route() dans le début de notre fichier python
- ☐ Avec app.route() au-dessus d'une méthode
- ☐ Avec app.run()

Question 9

Quelle sont les types de données que l'on peut envoyer dans un lien ?

- ☐ Float, number, boolean, string
- ☐ String, int, float, path
- ☐ Float, number, path, string

Question 10

Laquelle des routes suivantes n'est pas correcte ?

- ☐ /accueil/
- ☐ /accueil
- ☐ /accueil/#client nom

## Solutions des exercices



**p. 6 Solution n°1**

Flask est un framework de développement Web créé en langage Python. Ce framework est basé sur la base solide du moteur de modèles Jinja et de la bibliothèque complète d'applications Web WSGI de Werkzeug.

**p. 6 Solution n°2**

Flask est utilisé pour créer des applications Web à l'aide du langage de programmation Python.

**p. 6 Solution n°3**

L'hôte par défaut du flaskon est un hôte local (127.0.0.1) et le port par défaut est 5 000.

**p. 6 Solution n°4**

L'hôte et le port par défaut de Flask peuvent être modifiés en modifiant les valeurs des paramètres d'hôte et de port lors de l'appel de la méthode run.

```
from flask import Flask
app = Flask(__name__)
@app.route("/")
def index():
    return "Hello, World!"
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8080)
```

**p. 11 Solution n°5**

Nous allons créer deux fichiers : un fichier python pour créer l'application Flask et un fichier HTML pour la page. Le fichier HTML ressemble à ceci :

```
1 HTML(< <html>
2
3   <body>
4     {% include 'navigation.html' %}
5 <p>Bonjour je m'appelle {{ nom }} </p>
6 </body>
7
8 </html>>
```

Le fichier Python de la création de l'application :

```
1 Python(<@app.route('/')
2 def bonjour():
3     nom = 'Laura'
4     render_template('bonjour.html', { 'nom': nom })>
```

### p. 13 Solution n°6

Nous allons utiliser un fichier python pour créer l'application Flask :

```
1 Python(<
2 @app.route('/addition/<int:nombre_1>/<int:nombre_2>')
3 def addition(nombre_1, nombre_2):
4     return nombre_1 + nombre_2
```

### p. 14 Solution n°7

Afin de réussir ce défi, il suffit de répliquer la logique de l'exercice d'application vu dans la partie du routage avec les différentes opérations.

```
1 Python(<
2
3 @app.route('/addition/<int:nombre_1>/<int:nombre_2>')
4 def addition(nombre_1, nombre_2):
5     return nombre_1 + nombre_2
6 @app.route('/soustraction/<int:nombre_1>/<int:nombre_2>')
7 def soustraction(nombre_1, nombre_2):
8     return nombre_1 - nombre_2
9 @app.route('/multiplication/<int:nombre_1>/<int:nombre_2>')
10 def multiplication(nombre_1, nombre_2):
11     return nombre_1 * nombre_2
12 @app.route('/division/<int:nombre_1>/<int:nombre_2>')
13 def division(nombre_1, nombre_2):
14     return nombre_1 / nombre_2
15 >
```

### Exercice p. 14 Solution n°8

#### Question 1


Comment peut-on activer le débogage dans Flask ?

- ☒ Passer la propriété debug avec la valeur True en paramètre dans la méthode run.
- ☐ On ne peut pas activer le débogage
- ☐ Dans app.route
- ☒ Cette ligne de code garantit le débogage : app.run(debug = True).

#### Question 2

À quoi l'activation de débogage dans Flask peut être utile ?


- ☐ L'activation de débogage augmente la sécurité en imposant un redémarrage manuel à chaque modification du code
- ☒ Le serveur se rechargera lorsque le code change et vous n'avez pas besoin de redémarrer manuellement après chaque modification du code
- ☐ L'activation de débogage active automatiquement les tests

 Quand le débogage est activé le serveur se charge à chaque modification de code.

### Question 3

Qu'est-ce que Jinja ?


- ☐ Il crée des éléments logiciels pour des fonctions
- ☒ C'est un moteur de template
- ☐ Il mappe les routes

 Jinja est un moteur de templates rapide, expressif et extensible. Des espaces réservés spéciaux dans le template permettent d'écrire du code similaire à la syntaxe Python. Ensuite, le template reçoit des données pour rendre le document final.

### Question 4

Quelle est la syntaxe correcte pour appeler la variable « *var* » dans du code HTML ?


- ☐ #var
- ☐ <<var>>.
- ☒ {{var}}

 Pour passer une variable dans le code HTML, on la passe entre {{ }}.

### Question 5

Quelle est la méthode qui permet d'afficher un contenu à partir d'une page HTML ?


- ☐ return\_page()
- ☒ render\_template()
- ☐ show\_content()

 render\_template() est la méthode qui permet d'appeler des pages HTML et les afficher.

### Question 6


Comment on peut charger un contenu HTML dans un autre contenu HTML ?

- ☒ Avec include
- ☐ Avec extend
- ☐ En concaténant les deux contenus

 Include demande à Jinja de charger un template séparé dans un autre template.


### Question 7

Comment on peut charger un contenu HTML qui étend un autre contenu HTML ?

- ☐ Avec include
- ☒ Avec extend
- ☐ En concaténant les deux contenus
-  Extend garantit que lorsque nous demandons une page template qui étend une autre, la page demandée va contenir tout le contenu HTML de la page qu'elle étend.


### Question 8

Comment on définit les routes en Flask ?

- ☐ Avec app.route() dans le début de notre fichier python
- ☒ Avec app.route() au-dessus d'une méthode
- ☐ Avec app.run()
-  En Flask app.route() ajoutée en dessus d'une fonction, permet de définir dans quelle route la fonction sera invoquée.


### Question 9

Quelle sont les types de données que l'on peut envoyer dans un lien ?

- ☐ Float, number, boolean, string
- ☒ String, int, float, path
- ☐ Float, number, path, string
-  En Flask, nous pouvons utiliser les convertisseurs string, int, float et path pour convertir la donnée dynamique récupérée depuis l'URL.

### Question 10

Laquelle des routes suivantes n'est pas correcte ?

- ☐ /accueil/
- ☐ /accueil
- ☒ /accueil/#client nom
-  Nous pouvons utiliser les deux syntaxes : /accueil ou /accueil/. L'utilisation de la première syntaxe renvoie une erreur 404, si nous essayons l'URL /accueil/. Tandis que la deuxième syntaxe redirige vers /accueil/ en cas de saisie de l'URL /accueil.