

Le Web

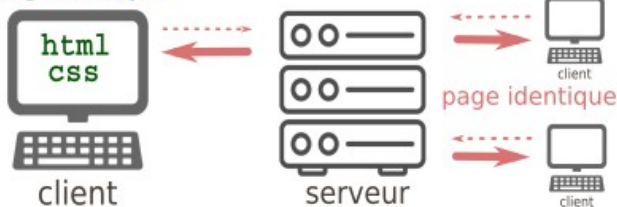
Relation client-serveur

Objectifs :

- Construire des formulaires HTML simples
- Construire quelques pages dynamiques simples sur un serveur
- Comprendre la différence entre les requêtes GET et POST

I Notion de page dynamique

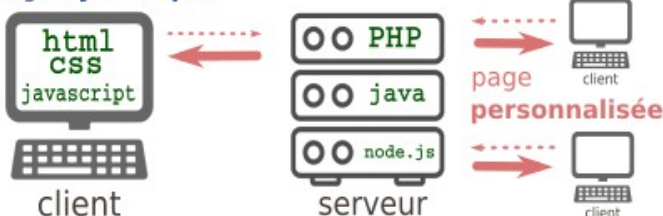
page statique



En abordant le HTML, les CSS et le JavaScript, nous avons vu ce qui se passait sur la machine client dans un navigateur. Si nous avons eu un serveur Web dit aussi serveur HTTP, il aurait envoyé la même page pour tous les utilisateurs.

Le document que nous aurait donné le serveur serait toujours le même à tout moment et pour tous les clients. Nous avons construit ce qu'on appelle des **pages statiques**.

page dynamique



Lorsque le contenu envoyé par le serveur dépend de ce que le client a demandé, on parle de **page dynamique**. Cela veut dire qu'en plus d'un serveur, il y a du code qui est exécuté côté serveur qui génère alors des pages différentes selon le client. C'est ce qui se

passé par exemple lorsque vous utilisez un moteur de recherche où la page affichée dépend des mots que vous avez recherchés. C'est aussi le cas lorsque vous vous connectez sur un site avec des identifiants. Le contenu des pages dépend de votre profil et des informations que vous avez données à ce site. Nous allons créer un petit serveur en Python, des langages classiques exécutés côté serveur sont PHP, java et node.js (qui permet d'utiliser du JavaScript côté serveur). Sur les sites professionnels en plus du serveur HTTP associé à un langage serveur, il y a souvent une base de données ce qui est au programme de terminale NSI.

II Un serveur en Python

À faire vous-même 1

Vous allez créer un serveur HTTP en Python.

Pour cela :

- Dans votre répertoire de travail créer un répertoire nommé « serveur » ;
- Recopiez le script suivant dans votre éditeur Python :

```
# -*- coding : utf-8 -*-
from flask import Flask
app = Flask(__name__)
@app.route('/')
def index():
    return "<p>Tout fonctionne parfaitement</p>"
app.run(debug=False)
```

- Enregistrez ce fichier dans le répertoire serveur en le nommant serveur.py ;
- Exécutez le script ;
- Dans un navigateur taper l'adresse : <http://127.0.0.1:5000/> ;
- Vous devriez voir la phrase « Tout fonctionne parfaitement » ;
- Si vous regardez le code source (Ctrl+U ou clic droit puis afficher le code) de cette page vous devriez obtenir : `<p>Tout fonctionne parfaitement</p>`.

Vous pouvez remarquer que:

- C'est exactement le contenu du return du script.
- Ce n'est pas une page HTML valide, mais votre navigateur l'affiche quand même.

- Retournez dans Spyder. Dans la fenêtre de la console vous devriez avoir :

```
* Serving Flask app "server" (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: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

- Arrêter le serveur dans Spyder en tapant Ctrl +C dans la console.

Explications

127.0.0.1 est adresse IP du serveur HTTP que vous avez créé sur votre machine vous auriez pu aussi taper localhost. C'est l'adresse de votre machine lorsque vous l'appellez depuis votre machine. 5000 est le port de votre serveur. L'adresse est donc de la forme :

$$\underbrace{\text{http://}}_{\text{Le protocole}} \underbrace{127.0.0.1}_{\text{l'adresse}} \underbrace{:5000}_{\text{le port}}$$

En exécutant le programme Python ci-dessus, le framework Flask a lancé un serveur web. Ce serveur web attend des requêtes HTTP sur le port 5000. En ouvrant un navigateur web et en tapant « 127.0.0.1:5000 », nous faisons une requête HTTP, le serveur web fourni avec Flask répond à cette requête HTTP en envoyant une page web contenant uniquement "<p>Tout fonctionne parfaitement</p>".

Expliquons le programme Python ligne à ligne.

```
from flask import Flask
```

Nous importons le module Flask.

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

Créer un objet Flask nommé app. C'est cet objet qui va contenir le serveur.

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

Créer un décorateur associé au chemin « / ». La notion de décorateur dépasse le programme de première NSI. Vous devez juste comprendre que lorsque le serveur reçoit une requête sur l'adresse « / » du serveur, celui-ci exécute la fonction qui suit ce décorateur. Ici la fonction index.

Cela veut dire que lorsque avec votre navigateur vous tapez 127.0.0.1:5000/ le serveur exécute la fonction index et retourne "<p>Tout fonctionne parfaitement</p>".

Nous verrons juste après comment ajouter une autre page à ce serveur.

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

Définis la fonction exécutée lorsque le serveur doit fournir la page correspondant à « / ».

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

Met en route le serveur.

À faire vous-même 2

- Dans Spyder exécuter le script serveur.py ;
- Dans votre navigateur taper l'adresse <http://127.0.0.1:5000/about> ;
- Vous devez obtenir une page dont le nom de l'onglet est « 404 Not found » c'est un message d'erreur 404 ce qui veut dire « ressource non trouvée » ;
- Éteignez votre serveur en tapant Ctrl +C dans la console de Spyder ;
- Complétez votre code afin d'obtenir le script suivant :

```
# -*- coding : utf-8 -*-  
from flask import Flask  
app = Flask(__name__)  
@app.route('/')  
def index() :  
    return "<p>Tout fonctionne parfaitement</p>"  
@app.route('/about')  
def about() :  
    return "<p> Une autre page Web </p>"  
app.run(debug=False)
```

- Exécuter ce code pour démarrer le serveur ;

- Dans votre navigateur taper l'adresse <http://127.0.0.1:5000/about/> ;
- Vous devriez obtenir une page affichant : « Une autre page Web ».

Votre serveur peut donc servir deux pages différentes. Cependant elles ne contiennent pas du code HTML valide et retourner une page valide serait un peu pratique car, il faudrait écrire tout le contenu de la page dans chaque fonction. De plus, une bonne pratique consiste à séparer au mieux le code HTML du code exécuter sur le serveur ici du Python. Flask comme les PHP, node.js, etc. permet d'utiliser un système de template (modèle en anglais) qui rend cela possible.

À faire vous-même 3

- Dans le répertoire serveur créez un répertoire templates.
- Dans ce répertoire créez un fichier index.html qui contient :

```
<! DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="auteur" content="NSI"/>
    <title>Serveur Flask Python</title>
  </head>
  <body>
    <h1>Mon site</h1>
    <p>Tout fonctionne parfaitement</p>
  </body>
</html>
```

- Modifier votre fichier serveur.py afin d'obtenir :

```
# -*- coding : utf-8 -*-
from flask import Flask, render_template
app = Flask(__name__)
@app.route('/')
def index():
    return render_template("index.html")
app.run(debug=False)
```

- Exécuter le script serveur.py ;
- Taper l'adresse 127.0.0.1:5000 dans votre navigateur ;
- Vous devriez voir s'afficher la page index.html.

En effet la ligne :

```
return render_template("index.html")
```

Demande au serveur de renvoyer le contenu du fichier index.html.

Le HTML est maintenant séparé du code Python, mais la page n'est pas dynamique au sens où c'est toujours la même qui sera servie.

À faire vous-même 4

Vous allez créer une page qui affiche l'heure du serveur.

- Dans le répertoire templates créer le fichier heure.html qui contient :

```
<! DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="auteur" content="NSI"/>
    <title>Serveur Flask Python</title>
  </head>
  <body>
    <h1>Mon site donne l'heure</h1>
    <p>Il est : {{heure}}h {{minute}} min {{seconde}} s</p>
  </body>
</html>
```

- Modifier le fichier serveur.py afin d'obtenir :

```
# -*- coding : utf-8 -*-
from flask import Flask, render_template
import datetime
app = Flask(__name__)
@app.route('/')
def index():
    return render_template("index.html")

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

app.run(debug=False)
```

- Exécuter le script serveur.py puis dans votre navigateur ouvrir la page
« <http://127.0.0.1:5000/heure> »
- Vous devriez obtenir une page qui affiche l'heure du serveur. À chaque fois que vous rafraichisserez la page (Ctrl + R) vous obtiendrez l'heure actuelle.

C'est la première page dynamique de votre serveur car à chaque requête vers cette page le contenu sera différent.

Explications :

```
import datetime
```

Importe le module datetime qui permet d'obtenir l'heure et la date du système.

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

Créer un objet date qui contient l'heure actuelle du système.

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

Créer des variables nommées h, m et s qui contiennent respectivement l'heure, les minutes et les secondes de l'heure du système.

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

La fonction `render_template` contient trois paramètres supplémentaires, ils se nomment `heure`, `minute` et `seconde`. Ils prennent pour valeur respectivement l'heure, les minutes et les secondes de l'heure courante du système.

Lorsque le serveur servira la page il remplacera les balise `{{heure}}`, `{{minute}}` et `{{seconde}}` par le contenu des variables `heure`, `minute` et `seconde`. C'est pourquoi lorsque vous faites une requête vers cette page vous ne verrez pas ces balises. S'il est 16:56:37 la balise `{{heure}}` sera remplacée par 16, la balise `{{minute}}` par 56 et la balise `{{seconde}}` par 37. Par contre, si vous ouvrez le fichier `heure.html` dans votre navigateur vous verrez bien les balises entre double accolade.

Ces balises ne sont pas des balises HTML, elles sont là pour être remplacées par des valeurs qui lui sont envoyées par la fonction `render_html`.

Un des grands intérêts de servir des pages dynamiques est de pouvoir traiter des formulaires HTML.

Nous allons donc maintenant voir comment faire un formulaire en HTML.

III Créer des formulaires en HTML

Conception d'un formulaire HTML

Un formulaire est une suite de champs que l'utilisateur peut remplir, puis envoyer vers une URL.

Un formulaire est compris entre des balises HTML `<form></form>`.

Des attributs de la balise `form` sont :

Attribut	Description
<code>action</code>	URL vers laquelle les informations soumises au formulaire sont envoyées.
<code>method</code>	La méthode HTTP utilisée pour les envoyer au serveur, pour nous cela sera GET ou POST.

Un formulaire simple :

```
<form action="" method="get">
<label for="GET-name">Nom :</label>
<input id="GET-name" type="text" name="name"/>
<button type="submit" >Envoyer</button>
</form>
```

Ici l'action est d'envoyer sur l'URL de cette page par la méthode GET.

Balise label et input

La balise auto fermante `<input />` permet de à l'utilisateur de saisir des données. Cette balise recueille des données qui peuvent se présenter sur la forme d'un champ texte, d'un bouton radio, d'une boîte à cocher, etc.

Il est généralement associé à un élément label qui permet de décrire ce que l'utilisateur doit saisir.

Attributs de la balise input :

Attribut	Description
<code>type="text"</code>	C'est une entrée de type texte.
<code>name</code>	Nom du champ associé à la valeur qui sera envoyée.
<code>id</code>	Identifiant dont la valeur doit être celle de l'attribut <code>for</code> du champ label pour les lier.
<code>value</code>	Optionnel : contient la valeur du champ.

Attributs de la balise label :

Attribut	Description
<code>for</code>	Doit être égal à l'id du champ input associé.

Champ email

Exemple :

```
<input name="mail" id="Mail" type="email" />
```

Après avoir saisi l'email, le champ vérifie automatiquement que c'est un email valide. En fait, il vérifie que le texte rentré est de la forme des caractères, « @ », des caractères, « . » des caractères. Sur les smartphones, cela permet d'afficher un clavier contenant le caractère « @ ».

Champ date

Exemple :

```
<input type="date" id="birthdate" name="datenaiss" />
```

Le champ doit être une date, le navigateur propose un menu permettant d'aider à entrer la date.

Bouton submit

Exemple :

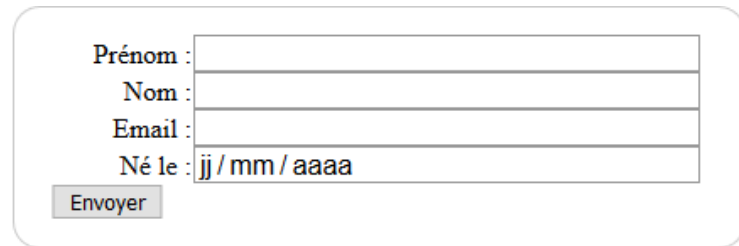
```
<button type="submit"> Envoyer </button>
```

Cliquez sur ce bouton déclenche l'envoi du formulaire vers l'URL de l'attribut `action` de la balise `form` et selon la méthode de l'attribut `method` de cette balise.

À faire vous-même 5

vous aller construire une page contenant ce formulaire :

Pour cela télécharger l'archive formulaire.zip.



The image shows a web form with four input fields and one button. The fields are labeled 'Prénom :', 'Nom :', 'Email :', and 'Né le :'. The 'Né le :' field has a date mask 'jj / mm / aaaa'. Below the fields is a button labeled 'Envoyer'.

- Elle contient un fichier HTML formulaireGet.html pré-rempli. Vous devrez remplacer les commentaires par les balises nécessaires.
- Elle contient un fichier CSS style.css qui est complet et que vous n'avez pas besoin de modifier. Vous pourrez néanmoins le modifier si vous le souhaitez après le travail demandé.
- Vous devez obtenir une page qui ressemble à la figure ci-dessus.

IV Requêtes GET et POST

On va modifier le script serveur.py afin d'avoir une page qui affiche les nom, prénom et email saisis dans le formulaire et affiche le nombre de jours depuis la date de naissance saisie.

À faire vous-même 6

Modifier la première ligne du script serveur.py afin d'obtenir :

```
from flask import Flask, render_template, request
```

Cela permet d'avoir accès à l'objet request qui permet d'obtenir les données des requêtes HTTP.

Ajouter à la fin du script juste avant le `app.run(debug=False)` :

```
@app.route("/age", methods=["GET"])
def age():
    annee, mois, jour = request.args["datenaiss"].split('-')
    annee = int(annee)
    mois = int(mois)
    jour = int(jour)
    dateNaissance = datetime.datetime(annee, mois, jour)
    aujourd'hui = datetime.datetime.today()
    a = aujourd'hui - dateNaissance
    n = request.args["nom"]
    p = request.args["prenom"]
    m = request.args["mail"]
    return render_template('age.html', nom = n, prenom = p, age = a.days, email = m)
```

Cela permet de créer une nouvelle page qui sera accessible à l'adresse « /age ». Il est supposé que

cette URL est accédée par une requête GET avec les paramètres nom, prenom, mail et datenaiss
S'il manque un des paramètres, vous obtiendrez un message d'erreur.

Dans un peu plus de détail :

```
@app.route("/age", methods=["GET"])
```

Créer une page accessible à l'adresse « /age » qui accepte la méthode GET.

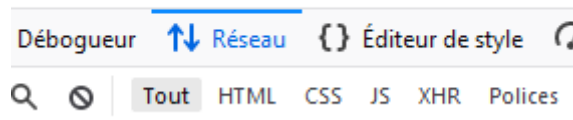
Dans la fonction age il faut comprendre que request.args est un dictionnaire Python qui a pour clés le nom des paramètres GET et les associe aux valeurs entrées dans le formulaire. Les valeurs sont des chaînes de caractères. Le module datetime permet de faire des calculs sur les dates ce qui permet d'obtenir le nombre de jours entre la date actuelle et la date saisie dans le formulaire.

Dans le répertoire templates ajouter le fichier HTML age.html dont le contenu est :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="auteur" content="NSI"/>
    <title>Serveur Flask Python</title>
  </head>
  <body>
    <h1>Bonjour {{prenom}} {{nom}} </h1>
    <p>Votre email est {{email}} et vous êtes né(e) depuis {{age}} jours</p>
  </body>
</html>
```

À faire vous-même 7

1. Exécuter le script server.py ;
2. Ouvrir dans un navigateur le fichier HTML formulaireGet.html
3. Taper Ctrl+ Maj + E pour obtenir l'outil réseau et cliquez pour sélectionner « Tout ».



4. Le compléter et cliquez sur le bouton « Envoyez ».
5. Constatez le résultat. Vous devriez voir le résultat de votre requête ainsi que l'ensemble des requêtes faites à votre serveur.

Analyse des requêtes GET

1. Combien de requêtes HTTP ont été effectuées lorsque vous vous êtes connecté au serveur ?
Quelle est la méthode utilisée pour chacune d'entre elles ?
2. Sur quel domaine ont été effectuées ces requêtes ?
3. Voici un tableau récapitulant les états des réponses des requêtes. Cette liste reprend les principales réponses du serveur lorsqu'il reçoit une requête du client. Vous trouverez la liste complète dans les liens de la partie « Pour en savoir plus ».
Quels sont les états des réponses aux requêtes réalisées sur le serveur web ?

Code	Message	Signification
200	OK	Requête traitée avec succès.
304	Not Modified	Document non modifié depuis la dernière requête.
404	Not Found	Ressource non trouvée.
500	Internal Server Error	Le serveur a rencontré une situation qu'il ne sait pas traiter.

4. Regardez dans la barre de navigation. Qu'observez-vous dans l'URL ?
5. Dans l'onglet Réseau cliquez sur la première requête HTTP en partant du haut (celle permettant d'accéder au document html) et observez le contenu de l'onglet En-têtes
 - * Où sont placées les données collectées par le formulaire dans l'URL de la requête ?
 - * Comment ces données sont-elles séparées ?
6. Vous pouvez visualiser les **en-têtes bruts** des requêtes faites par le client et des réponses données par le serveur. Recherchez :
 - * La version du navigateur utilisé par le client ainsi que son système d'exploitation ;
 - * La version du protocole HTTP utilisée (*HTTP/n*) ;
 - * Le logiciel utilisé par le serveur (*server* ou *X-Powered-By*).
7. Observez dans la console de Spyder ce qui a été envoyé par le serveur.
8. Modifier des valeurs, par exemple la valeur du nom, dans l'URL
/!\ Pas les noms de paramètres sinon vous obtiendrez une erreur 500.
Qu'observez-vous ?

Vous pouvez maintenant éteindre le serveur en tapant Ctrl + C dans la console Python.

Bilan :

La méthode GET sert pour demander des ressources au serveur typiquement un document ou une requête sur un moteur de recherche. Lors d'une requête GET, à la fin de l'URL de l'adresse Web, il y a un point d'interrogation (?) suivi par les paires nom/valeur séparées par une esperluette (&). Les paramètres sont donc visibles directement dans l'URL. Une requête GET envoyant les paramètres param1 de valeur valeur1 et param2 de valeur valeur2 se terminera par :

`?param1=valeur1¶m2=valeur2`

La requête GET ne doit pas être utilisée pour des données sensibles car, elles sont visibles dans l'URL. Comme vous l'avez constaté dans la console Python, les paramètres sont dans les traces de

la connexion ce qu'on appelle les log. De plus, la plupart des serveurs imposent une taille maximale aux URL, on ne peut donc envoyer des données dont la taille serait importante. Néanmoins la méthode GET est la méthode d'accès par défaut à une URL. C'est cette méthode que vous utilisez lorsque accéder à un site à la première page d'un site.

Analyse des requêtes POST

1. Enregistrez le fichier HTML formulaireGet.html sous le nom formulairePost.html. Puis modifier l'attribut method de la balise form pour que le formulaire soit envoyé selon la méthode POST. Votre balise form devrait ressembler à ceci :

```
<form id="monFormulaire" method="POST" action="http://127.0.0.1:5000/age" >
```

Enregistrez le fichier formulairePost.html.

2. Modifier le fichier server.py afin qu'il accepte aussi les requêtes POST. Il n'y a que la partie concernant la page "/age » à modifier :

```
@app.route("/age", methods=["GET", "POST"])
def age():
    if len(request.args) > 0 :
        annee, mois, jour = request.args["datenaiss"].split('-')
        annee = int(annee)
        mois = int(mois)
        jour = int(jour)
        dateNaissance = datetime.datetime(annee, mois, jour)
        aujourd'hui = datetime.datetime.today()
        a = aujourd'hui - dateNaissance
        n = request.args["nom"]
        p = request.args["prenom"]
        m = request.args["mail"]
    else :
        annee, mois, jour = request.form["datenaiss"].split('-')
        annee = int(annee)
        mois = int(mois)
        jour = int(jour)
        dateNaissance = datetime.datetime(annee, mois, jour)
        aujourd'hui = datetime.datetime.today()
        a = aujourd'hui - dateNaissance
        n = request.form["nom"]
        p = request.form["prenom"]
        m = request.form["mail"]
    return render_template('age.html', nom = n, prenom = p, age = a.days, email = m)
```

La ligne suivante permet au serveur d'accepter les requêtes POST pour la page "/age ».

```
@app.route("/age », methods=["GET", "POST"])
```

Le test suivant permet de choisir de lire les paramètres GET si la longueur du dictionnaire contenant les paramètres GET est non vide sinon il lit les paramètres POST.

`if len(request.args) >0 :`

Vous pouvez remarquer que la différence principale de la suite est que l'on utilise le dictionnaire `request.args` pour la méthode GET et le dictionnaire `request.form` pour la méthode POST.

3. Démarrez le serveur, ouvrez la page HTML `formulairePost.html` dans votre navigateur et ouvrez le panneau réseau (Ctrl+ Maj + E) en sélectionnant « Tout ».
4. Remplissez le formulaire et envoyez-le.
5. Regardez dans la barre de navigation. Comparez l'URL de la méthode POST avec celle de la méthode GET.
6. Cliquez dans l'onglet Réseau sur la première requête http en partant du haut et regardez à droite dans « En têtes ».
* Quelle est la méthode de la requête ?
Expliquez pourquoi la méthode POST est recommandée pour l'envoi d'un formulaire avec des données sensibles ?
7. Cliquez sur l'onglet « paramètres ». Que constatez-vous ? Peut-on dire que les données sont sécurisées ?

Vous pouvez maintenant éteindre le serveur en faisant Ctrl + C dans la console Python.

Bilan :

La méthode POST n'a pas de limite dans la taille des données transmises au serveur. Les données du formulaire sont dans l'entête de la requête. Les données ne sont donc pas visibles dans l'URL, c'est pourquoi la méthode POST est recommandée pour les données sensibles (mot de passe par exemple) et pour modifier des données sur le serveur. Cela ne suffit pas à sécuriser l'envoi des données. Il faut de plus que la connexion entre le client et le serveur le soit, pour cela il faut utiliser le protocole HTTPS (HTTP Secure) qui crypte les données entre le client et le serveur, ce qui assure que seul le serveur aura accès aux données et non pas les machines qui ont pu servir de relais entre le client et le serveur.

Serveur de tests en ligne :

Pour tester vos formulaires vous pouvez remplacer `127.0.0.1:5000` par :

<https://flask-get-post.glitch.me>

Si vous voulez tester d'autres formulaires utilisant les méthodes GET ou POST vous pouvez utiliser l'adresse suivante qui affichera tous les paramètres que vous lui aurez transmis :

<https://flask-get-post.glitch.me/form>

Pour en savoir plus :

- Tutoriel Flask :
<http://sdz.tdct.org/sdz/creez-vos-applications-web-avec-flask.html>
<https://openclassrooms.com/fr/courses/4425066-concevez-un-site-avec-flask>
- La balise form : <https://developer.mozilla.org/fr/docs/Web/HTML/Element/Form>
- Construire son premier formulaire :
https://developer.mozilla.org/fr/docs/Web/Guide/HTML/Formulaires/Mon_premier_formulaire_HTML
- La balise input : <https://developer.mozilla.org/fr/docs/Web/HTML/Element/Input>
- Code de réponse entête HTTP : <https://developer.mozilla.org/fr/docs/Web/HTTP/Status>
- A propos des requêtes GET et POST
https://developer.mozilla.org/fr/docs/Web/Guide/HTML/Formulaires/Envoyer_et_extraire_les_donnees_des_formulaires
<https://www.xul.fr/ecmascript/get-post.php>
- Les autres méthodes du protocole HTTP :
<https://developer.mozilla.org/fr/docs/Web/HTTP/Méthodes>

Sources :

- https://pixees.fr/informatiquelycee/n_site/nsi_prem_flask.html
- https://framagit.org/patrice.thibaud/nsi_1ere_lmdf/-/blob/master/docs/12_reseaux/c12_p1_http/tp_formulaire/tp_formulaire.md

Accéder au code en ligne :

<https://glitch.com/edit#!/flask-get-post>

Il est un peu différent de celui de ce TP. J'ai dû ajouter un module qui permet d'afficher l'heure dans notre fuseau horaire et j'ai ajouté une feuille CSS.