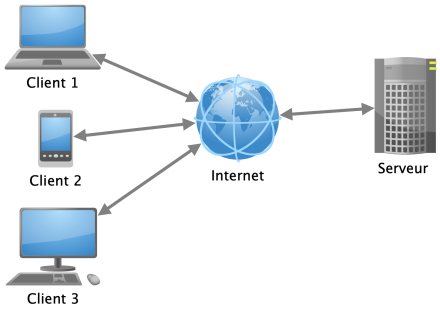
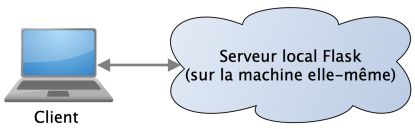
Interaction entre l'homme et la machine sur le web

Interaction client-serveur avec Flask

Précédemment nous avons créé un formulaire. Celui-ci ne fonctionne pas car il essaie d'envoyer les données collectées sur un site qui n'est pas le bon.
Lorsqu'un utilisateur surfe sur le Web avec un ordinateur (une tablette ou un smartphone), les interactions avec un site se passent comme sur le schéma ci-contre : le site consulté est hébergé sur un *serveur distant* auquel on accède grâce à son URL (*Uniform Resource Locator*, l'adresse du site). Plus le nombre de consultations simultanées du site est grand, plus le serveur doit-être puissant pour pouvoir répondre aux demandes et traiter les informations.





Nous allons simuler la situation précédente sans recourir à un serveur distant : nous allons créer un *serveur local*. Dans beaucoup de situations, le serveur (local ou distant) produit des pages HTML et traite les informations en utilisant le langage PHP.
Cependant nous pouvons aussi nous servir de PYTHON pour faire la même chose, à l'aide de la bibliothèque `Flask` !

Pour travailler avec `Flask`, on va utiliser un *environnement virtuel* sous PYCHARM. Pour créer un tel environnement, regarde la vidéo suivante :

https://youtu.be/uyctvyt0a8s

Un exemple minimal de soumission de formulaire avec Flask

Fabriquons un site très simple : on accède à la page `index.html` et on remplit le formulaire, puis à l'envoi du formulaire, une page `resultat.html` est affichée, qui exploite la donnée soumise *via* le formulaire.

Il y a donc un fichier `index.html`.

Code HTML

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Exemple simple</title>
</head>
<body>
<form action="http://localhost:5000/resultat" method=post>
    <label for="ChampNom">Entre ton nom :</label>
    <input id="ChampNom" name="nom" type="text">
</form>
</body>
</html>
```

Tu peux remarquer que le formulaire envoie les données à http://localhost:5000, c'est à dire à lui-même (c'est ce que veut dire `localhost`) et sur le port 5000 (car `Flask` est configuré comme cela par défaut).

Voici le fichier `resultat.html` :

Code HTML

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Résultat</title>
</head>
<body>
Bonjour à toi, {{nom}} !
</body>
</html>
```

Il faut comprendre que `{{nom}}` fait référence la variable `nom` de la balise `<input>` de la page `index.html`.

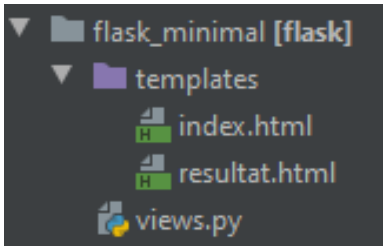
Il nous reste à connecter ces deux éléments en mettant en place un serveur `Flask` et c'est là que PYTHON intervient.

Code Python

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

# on importe Flask qui permet de créer le serveur
# render_template permet d'afficher des pages déjà crées
# en remplaçant chaque élément noté {{element}} par la valeur
# de la variable element correspondants


app = Flask(__name__)  # on crée le serveur


@app.route('/')  # on définit ce qui se passe à la connexion
def index():
    return render_template("index.html")


@app.route('/resultat', methods=['post'])  # que se passe-t-il pour
    resultat ?
# on traite un formulaire avec la méthode POST
def resultat():
    result = request.form  # on récupère les infos du formulaire
    # sous la forme d'un dictionnaire Python dont les clés
    # sont les noms des variables.
    valeur_nom = result['nom']
    return render_template("resultat.html", nom=valeur_nom)


app.run(debug=True)  # on démarre le serveur
```

Pour coordonner tout cela il faut

- Créer un répertoire pour mettre le projet (j'ai appelé le mien `flask_minimal`);
- créer le fichier PYTHON `views.py`;
- créer un répertoire appelé obligatoirement `templates`;
- créer dans ce répertoire les fichiers `index.html` et `resultat.html`.



À toi de jouer

Tu vas reprendre le formulaire que tu as crée à l'activité « produire un formulaire » et t'en servir comme base d'un petit projet `Flask`.

Exercice 1

Crées une application web avec `Flask` qui utilise ton formulaire comme point de départ : tu pourras renommer cette page `index.html`. Il faudra bien veiller à modifier la balise `<form>` comme ceci :

```
<form action="http://localhost:5000/resultat" method=post>
```

Ensuite tu pourras par exemple analyser la date de naissance de l'utilisateur **dans le fichier views.py** et en fonction du résultat afficher une page `mineur.html` ou bien `majeur.html`.
C'est à toi d'imaginer comment à l'intérieur du fichier `views.py` tu peux traiter les informations du formulaire pour dynamiser ton exemple de site...