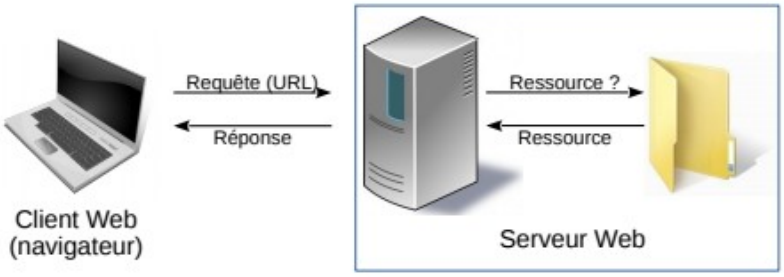
MINIPROJET 4 Jouer en réseau en python

Un serveur web est une application spécifique qui tourne sur un ordinateur. Il existe des serveurs écrit en langage C, en Java, en Python,…

Le rôle du serveur Web est de servir des clients. Ces derniers envoient au serveur une requête précisant leur demande. Le serveur répond à cette demande si celle-ci est valide.

Pour communiquer le serveur et le client utilise un langage commun, il s'agit du protocole **HTTP**.



Les échanges entre le client et le serveur se font sous forme de **paquets** qui transportent sur le réseau Internet les requêtes du client et les réponses du serveur.

Ce langage est assez compréhensible pour nous, voici par exemple la requête d'un client demandant la page **1NSI.html** sur le serveur **daldegan.yo.fr** :

```
GET /1NSI.html HTTP/1.0
Host: daldegan.yo.fr
```

Cette requête correspondrait à la saisie par un utilisateur de l'**URL** :
https://daldegan.yo.fr/1NSI.html

Ici **daldegan.yo.fr** correspond au nom de la machine, on dit aussi **nom de domaine** de la machine. Pour trouver leur chemin sur le réseau Internet ces paquets ont besoin de connaître l'**adresse IP** du serveur. Pour trouver cette adresse IP à partir du nom de domaine le client interroge un **serveur DNS**. Ici, notre serveur **daldegan.yo.fr** aurait par exemple pour adresse IP **146.88.237.17**

Note : Pour connaître l'adresse IP d'un serveur, on peut passer par un service de DNS lookup (ou annuaire inversé), mais plus simplement, en terminal :

```
$ host nom_de_mon_serveur
```

ETAPE 1 : Un premier serveur

Une façon simple et légère de créer un serveur web avec python.

1) Copier sur le bureau les fichiers ***index.html*** et ***serveur1.py***.

2) Donner les droit d'exécution pour ***serveur1.py*** à l'aide de la commande :

```
$ chmod +x serveur1.py
```

3) Démarrer ***serveur1.py*** dans un terminal.

```
$ python3 serveur1.py
```

Si le message suivant apparaît,

```
OSError: [Errno 98] Address already in use
```

modifier la valeur de la variable *PORT*, par exemple 8084.

4) Taper l'URL suivante dans le navigateur **http://localhost:8080/index.html**
Essayer ensuite avec **127.0.0.1:8080/index.html**

5) Modifier la page web ***index.html*** et afficher de nouveau la page.

Contenu du fichier ***serveur1.py***

```
#!/usr/bin/python3                                  → indique le programme à utiliser pour l'exécution directe



import http.server                                  → bibliothèques à importer pour gestion d'un serveur web http
import socketserver

PORT = 8080                                         → définition du port utilisé
handler = http.server.SimpleHTTPRequestHandler      → définition du serveur de requêtes


httpd = socketserver.TCPServer(("", PORT), handler) → création de l'instance du serveur

print("Serveur opérationnel sur le port : ", PORT)

httpd.serve_forever()                               → démarrage du serveur
```

6) Se connecter sur les autres serveurs disponibles du réseau.

On pourra utiliser une des commandes ci-dessous pour déterminer les adresses des machines sur le réseau :

Pour sa machine	`$ ifconfig`
Pour le réseau	`$ nmap xxx.xxx.xxx.0/24`

ETAPE 2 : Un serveur avec formulaire

Dans cette étape nous allons voir comment rendre le serveur interactif avec le module **cgi** et aussi que l'on peut intégrer le code html directement dans le fichier python.
(on pourra voir https://info.blaisepascal.fr/nsi-serveur-http-python-cgi)

1) Créer dans le même répertoire les fichiers **serveur2.py** et **formulaire.py** suivants :

serveur2.py

```
#!/usr/bin/env python3

import http.server

PORT = 8888
server_address = ("", PORT)

server = http.server.HTTPServer
handler = http.server.CGIHTTPReques-
tHandler
handler.cgi_directories = ["/"]
print("Serveur actif sur le port :", PORT)

httpd = server(server_address, handler)
httpd.serve_forever()
```

formulaire.py

```
#!/usr/bin/env python3
# coding: utf-8

import cgi

form = cgi.FieldStorage()
print("Content-type: text/html; charset=utf-8\n")

print(form.getvalue("name"))

html = """<!DOCTYPE html>
<head>
   <title>formulaire</title>
</head>
<body>
   <form action="/formulaire.py" method="post">
      <input type="text" name="name" value="Votre
nom" />
      <input type="submit" name="send" value="En-
voyer information au serveur">
   </form>
</body>
</html>
"""
print(html)
```

2) Modifier les permissions pour **formulaire.py**, car il doit maintenant pouvoir être exécuté par la machine. Taper dans un terminal :

```
$ chmod +x formulaire.py
```

(ce n'est pas obligatoire pour **serveur2.py**)

3) Taper l'URL suivante dans le navigateur : **localhost :8888/formulaire.py**

4) Saisir des informations dans le formulaire et observer les informations transmises par le serveur en modifiant l'attribut **method** :

`method=''post''` ou `method=''get''`

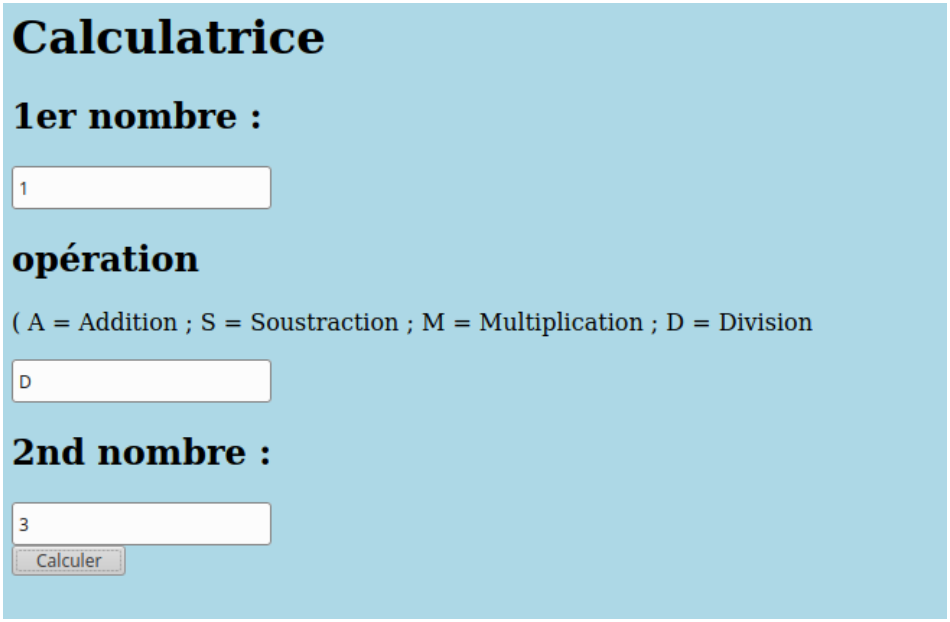
Expliquer ce que signifie « GET » et « POST ».

5) Modifier le code précédent pour afficher un style différent, des images, d'autres messages…

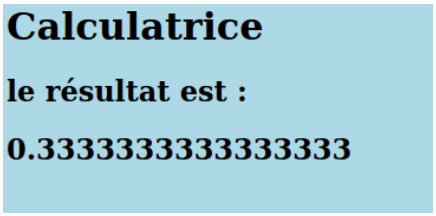
ETAPE 3 : Une calculatrice

1) Modifier le fichier **formulaire.py** pour permettre de faire une calculatrice *« en ligne »*, pour avoir un rendu de ce type :

formulaire.py



Le bouton *Calculer* du fichier **formulaire.py** *appelle* le script **calcul.py** qui permet le calcul :



Créer ce fichier **calcul.py.**

2) Utiliser la calculatrice d'autres élèves sur le réseau.

3) **BONUS :** Améliorer cette calculatrice pour obtenir ce genre de résultat, avec une seule saisie :

ETAPE 4 : Le jeu

Il s’agit maintenant d’adapter un jeu très simple « deviner un nombre » pour l’implémenter sur un serveur python afin de pouvoir jouer en ligne sur le réseau !

- Développer une interface web avec un serveur **serveur4.py** et la relier au programme fourni **devine.py** pour saisir les nombres testés uns à uns avec affichage des nombres saisis au fur et à mesure.

COMPETENCES TRAVAILLEES

APP	Mobiliser les concepts et les technologies adaptés au problème
REA	Imaginer et concevoir une solution, décomposer en blocs, se ramener à des sous-problèmes simples et indépendants, adopter une stratégie appropriée

ANNEXE : Problèmes de ports

Lors de lancement multiples de serveurs, en changeant régulièrement les numéros de ports (comme proposé plus haut), on peut être perdu sur les ports utilisés.
Pour en avoir une idée, il faut repérer les ports utilisés par le protocole TCP en mode « écoute » (*LISTEN*).

La commande suivante donnera une idée :

```
$ sudo lsof -nP -i tcp | grep LISTEN
```

Lorsqu'un serveur python est démarré, la dernière ligne est par exemple :
*python3 6165 blanche 3u IPv4 193516 0t0 TCP *:8080 (LISTEN)*

Ce qui signifie que l'utilisateur « blanche » a bien lancé un serveur TCP python, qui est en attente sur le port 8080.

Pour fermer ce serveur, il suffit de « tuer » le processus en cours :

```
$ kill 6165
```

D'autres explications sur cette commande ici : https://debian-facile.org/doc:systeme:lsof