

Programmation Python

(adoptez un serpent !)

Programmation web

Pr Amadou Dahirou GUEYE

dahirou.gueye@uadb.edu.sn

Université Alioune Diop

Master2 SI/SR

Octobre 2022

Sommaire

- Serveur HTTP simple
- CGI
- Serveur HTTP autonome
- Site web dynamique avec WSGI
- TP sous Django
- Projet

Dialogue au Resto : client - serveur

Garçon, un Jus de citron !

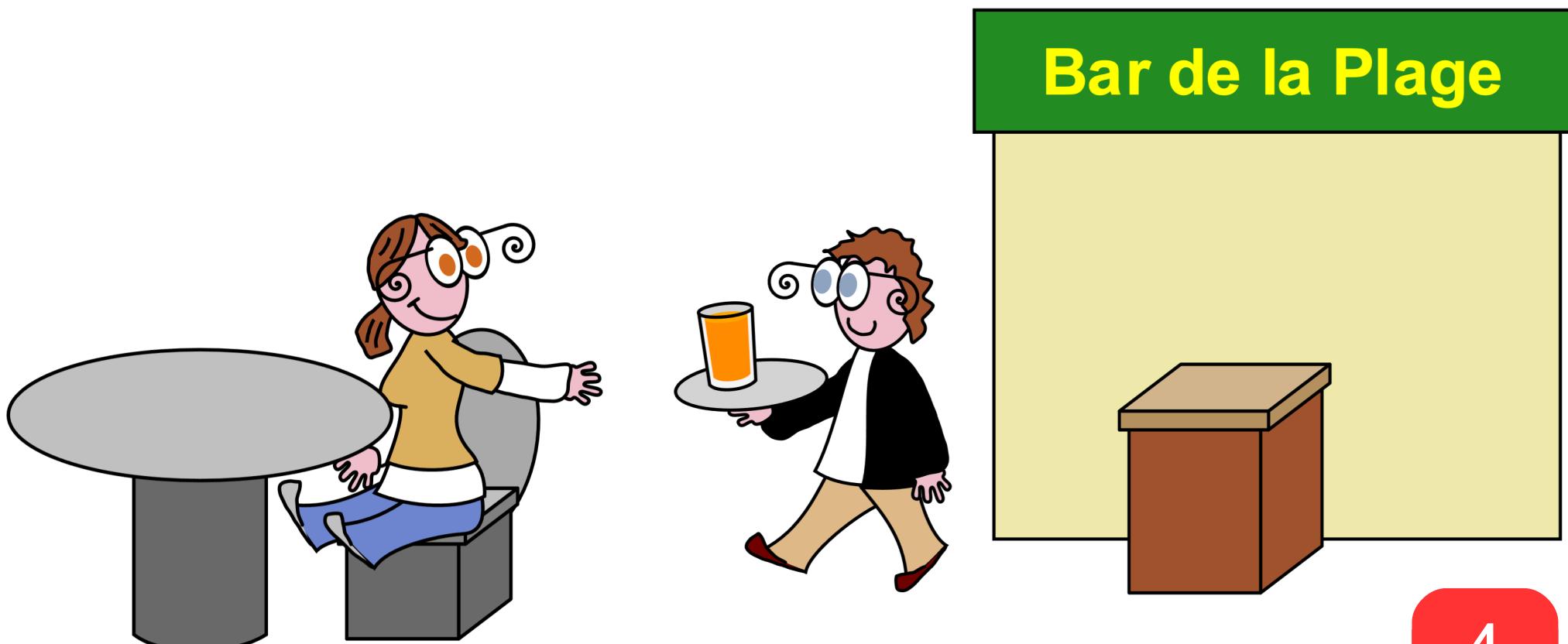


Bar de la Plage



Dialogue au Resto : client - serveur

Voilà voilà !



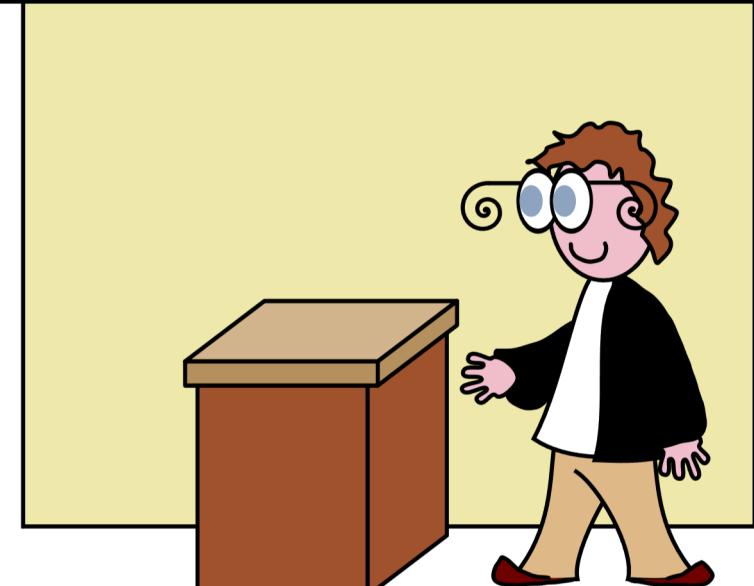
Dialogue au Resto : client - serveur

Garçon, la page web

http://bar_de_la_plage.fr/jus_citron.html



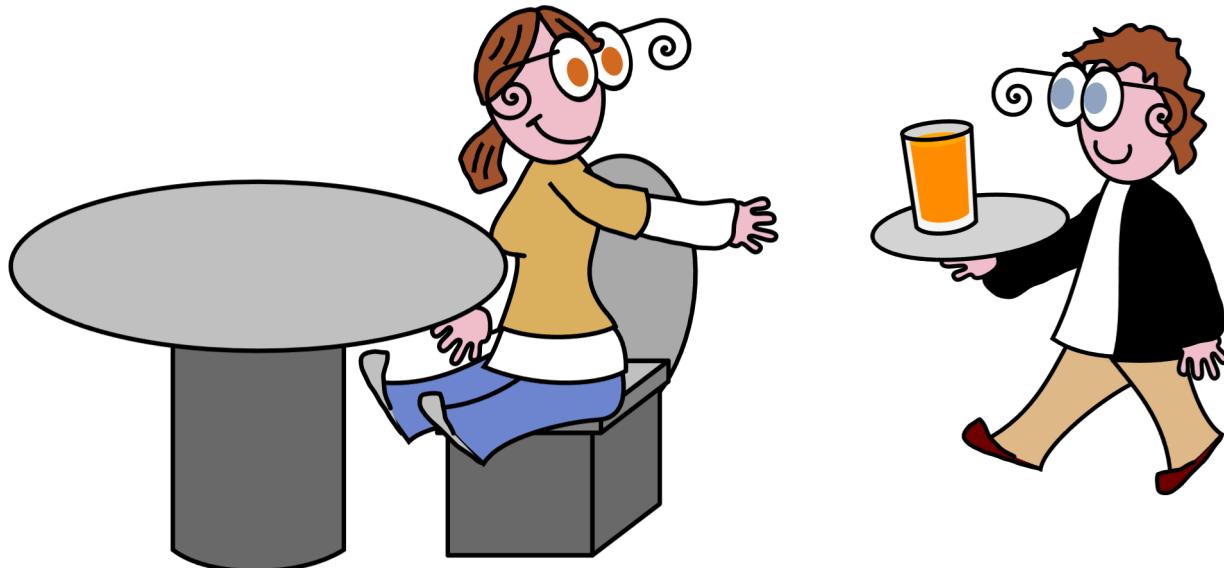
Bar de la Plage



Dialogue au Resto : client - serveur

Voilà voilà :

```
<html>
<head><title>Bar de la plage</title></head>
<body>
Notre spécialité, le jus de citron !
</body>
</html>
```



Bar de la Plage

Le protocole HTTP : client - serveur

GET http://bar_de_la_plage.fr/jus_citron.html HTTP/1.0

User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:46.0) Gecko/20100101 Firefox/46.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.5,en;q=0.3

Accept-Encoding: gzip, deflate

Connection: keep-alive

Requête HTTP

avec des informations sur les formats acceptés, la langue préférée, le support de la compression, etc.



Bar de la Plage



Le protocole HTTP : client - serveur

HTTP/1.0 200 OK

Content-Type: text/html; charset=utf-8

Date: Thu, 02 Jun 2016 12:49:35 GMT

Server: BaseHTTP/0.6 Python/3.5.1

content-length: la taille du message en octets

cache-control: (option max-age=<seconds>)

```
<html>
<head><title>Bar de la plage</title></head>
<body>
Notre spécialité, le jus de citron !
</body>
</html>
```

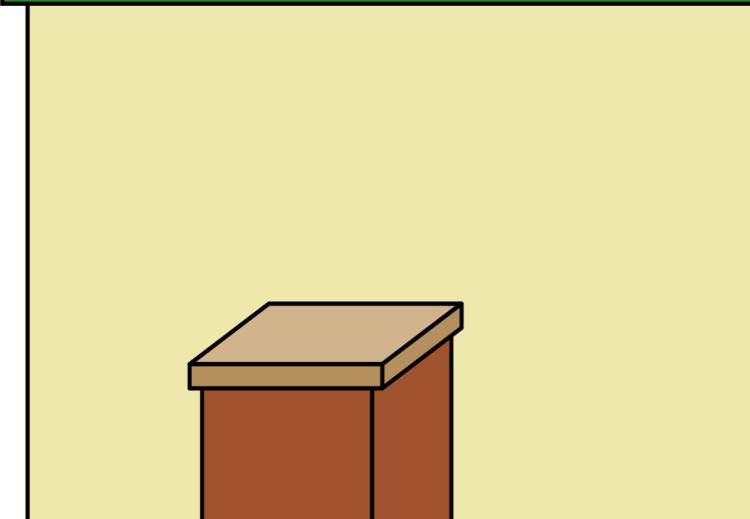


Réponse HTTP

avec des informations sur le format et l'encodage du fichier, la date,...

Une ligne vide sépare l'enveloppe HTTP du contenu

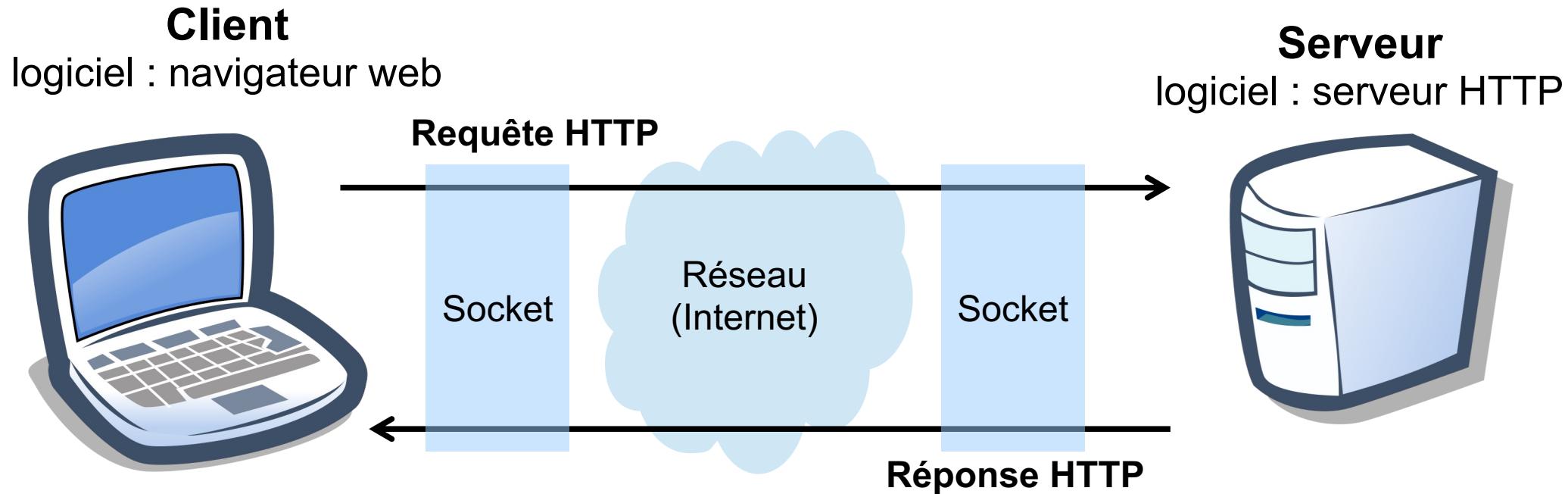
Bar de la Plage



Remarque sur les en-têtes de réponse

- **content-type**: le type de contenu de la réponse (au format MIME). Exemple: **text/html**, **image/png**, **application/pdf**.
- **content-length**: la taille du message en octets.
- **cache-control**: permet d'influer sur le comportement des caches, notamment pour indiquer la durée de validité maximale (option max-age=<seconds>).
- Les en-têtes **content-type** et **content-length** sont en fait utilisés pour tout message ayant un contenu, y compris certaines requêtes (e.g. POST).

Le protocole HTTP : client - serveur



Une requête par fichier à récupérer (pas de commande groupée !)

Exemple :

- + une requête pour la page HTML
- + une requête pour la feuille de style CSS
- + une requête pour chaque image JPG, PNG ou SVG

Les commandes HTTP

- **GET** : obtenir le document à l'adresse demandée
- **POST** : envoie de données ou d'un document (sans adresse précise)
- **HEAD** : obtenir l'en-tête du document à l'adresse donnée
- **PUT** : ajout d'un document à l'adresse donnée
- **DELETE** : suppression du document à l'adresse donnée

- Seules les commandes **GET**, **POST** et **HEAD** sont fréquemment utilisées

Programmation web en Python

Serveur HTTP simple

Mise en place d'un place serveur simple http avec python

- Pour mettre en place un serveur, nous utilisons python à la place de Apache et PHP. Autrement dit, des modules natifs de python
- Coté client: HTML et CSS pour faire nos pages web, le design, etc.
- Nous allons utiliser
 - Le module Python `http.server` qui contient un serveur HTTP déjà implémenté et prêt à l'emploi
 - Le module `socketserver` pour faire communiquer deux applications via un réseau
 - La classe `SimpleHTTPRequestHandler` qui est un gestionnaire de requête http
 - La classe `TCPServer` du module `socketserver` pour établir la connexion

Mise en place d'un serveur standard http avec python

■ Créer un fichier http-server.py

```
#coding:utf-8
```

```
import http.server
```

```
import socketserver
```

```
port=80
```

```
address=("","",port)
```

```
handler=http.server.SimpleHTTPRequestHandler
```

```
httpd=socketserver.TCPServer(address,handler)
```

```
print(f"Serveur démarré le 28 juillet 2021 sur le port {port}")
```

```
httpd.serve_forever() # desservir le serveur de manière continue
```

Comment le démarrer? Il faut ouvrir un terminal

Fonctionnement du serveur par la création d'une page HTML

- Créer un fichier index.html à partir du répertoire public qui va constituer la racine de notre page web

```
<!DOCTYPE html>

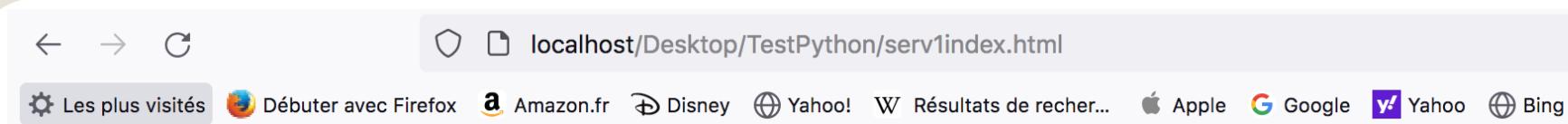
<html lang="fr">

<head>
    <meta charset="utf-8">
    <title>Une page web avec python et HTML!</title>
</head>

<body>
    <h1>Bienvenue chers Etudiants Master SI/SR 2020</h1>
    <p>Je suis en python, avec une page web HTML</p>
</body>

</html>
```

Fonctionnement du serveur par la création d'une page HTML



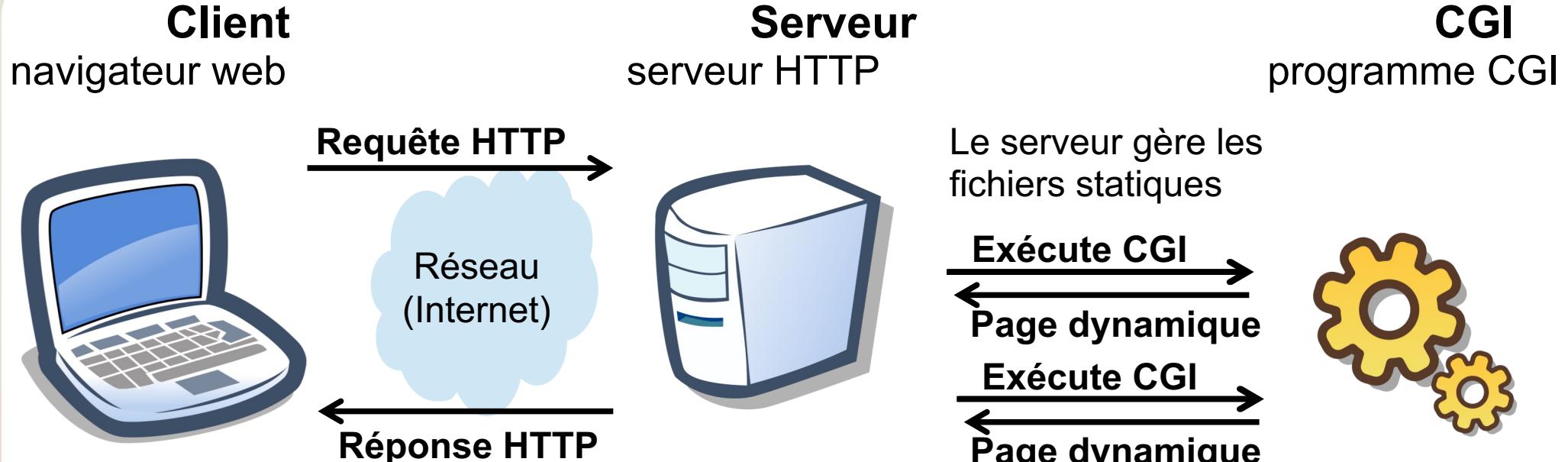
Bienvenue chers Etudiants Master SI/SR Juillet 2021

Je suis en python, avec une page web HTML

```
[bash-3.2# python3 /Users/macbookpro/Desktop/TestPython/serv1.py
Serveur démarré le 28 juillet 2021 sur le port 80
127.0.0.1 -- [27/Jul/2021 15:37:41] "GET /Desktop/TestPython/serv1index.html HT
TP/1.1" 200 -
```

- On peut à partir de là mettre du CSS d'autres pages web pour communiquer chacune avec l'autre.
- Cependant avec ce qu'on a mis en place, on ne peut faire que du HTML, donc c'est du statique
- Par défaut, le serveur http ne fait que télécharger le fichier HTML et lit le contenu pour être interprété mais n'exécute pas un programme

CGI



■ CGI (Common Gateway Interface)

- Le serveur gère les pages et les fichiers statiques
- Le programme CGI est exécuté pour chaque page dynamique, et retourne le résultat
 - Par rapport à l'approche serveur HTTP autonome : le CGI sépare les fichiers statiques (optimisé) et les pages dynamiques
 - Problème de performance : le programme CGI doit être exécuté une fois par page, en particulier si le temps de lancement est long !

Interface CGI

■ Maintenant, si on veut faire communiquer plusieurs fichiers python, il va falloir programmer et donc exécuter un script

■ Utiliser les interfaces CGI:

- Interface haut niveau qui va se charger d'exécuter le programme et de retourner le résultat
- À exécuter sous forme de script (ça peut être un script en php, ou en python, ou en C)
- Indépendant du langage utilisé

■ Pour cela, il faut donc modifier notre serveur.

Serveur fonctionnant en mode CGI

1. `#coding:utf-8`
2. `import http.server`
3. `port=80`
4. `address=("",port)`

5. `server=http.server.HTTPServer # démarrage du serveur`
6. `handler=http.server.CGIHTTPRequestHandler # fonctionnant avec l'interface de script cgi`
7. `handler.cgi_directories=["/"] # qui prend une liste de répertoires où il va devoir chercher vos scripts (la racine)`

8. `httpd=server(address, handler) # Création du serveur`

9. `print(f"Serveur CGI démarré sur le port {port}")`

10. `httpd.serve_forever()`

Utilisation du langage python pour la création de page web

Créer un fichier index.py à la racine de votre projet (Premier exemple)

```
1. #coding:utf-8  
2. import cgi  
3. print("Content-type: text/html; charset=utf-8 \n")  
4. html="""<!DOCTYPE html>  
5. <head>  
6.   <meta charset="utf-8">  
7.   <title>Ma page web!</title>  
8. </head>  
9. <body>  
10.  <h1>Bonjour</h1>  
11.  <p>Master SI/SR Juillet 2021</p>  
12. </body>  
13. </html>  
14. """  
15. print(html)
```

Utilisation du langage python pour la création de page web

```
[bash-3.2# python3 /Users/macbookpro/Desktop/TestPython/index.py
Content-type: text/html; charset=utf-8

<!DOCTYPE html>
<head>
    <meta charset="utf-8">
<title>Ma page web!</title>
</head>
<body>
    <h1>Bonjour</h1>
    <p>Script Master SI/SR Juillet 2021</p>
</body>
</html>

bash-3.2# ]
```

Utilisation du langage python pour la création de page web

A screenshot of a Firefox browser window. The address bar shows "localhost/Desktop/TestPython/index.py". Below the address bar, the toolbar includes links for "Les plus visités", "Débuter avec Firefox", "Amazon.fr", "Disney", "Yahoo!", "Résultats de recher...", "Apple", and "Google". The main content area displays the following Python code:

```
#coding:utf-8
import cgi
print("Content-type: text/html; charset=utf-8 \n")
html="""<!DOCTYPE html>
<head>
    <meta charset="utf-8">
    <title>Ma page web!</title>
</head>
<body>
    <h1>Bonjour</h1>
    <p>Script Master SI/SR Juillet 2021</p>
</body>
</html>
"""
print(html)
```

```
[bash-3.2# python3 /Users/macbookpro/Desktop/TestPython/serv1.py
Serveur CGI démarré sur le port 80
127.0.0.1 - - [27/Jul/2021 16:09:41] "GET /Desktop/TestPython/index.py HTTP/1.1"
200 -]
```

Limitations de CGI

- Il va démarrer une requête à chaque fois que vous allez faire une actualisation.
- A chaque changement, il démarre un nouveau processus pour exécuter la requête.
- Ce qui n'est pas optimisé pour de grosses applications
- Utiliser FastCGI ou WSGI
- FastCGI: un système d'interface comme CGI, sauf qu'au lieu d'exécuter à chaque fois un processus, il exécute qu'une seule fois et sauvegarde en mémoire. Et comme ça à chaque fois que vous avez besoin de faire une nouvelle requête, il récupère ce qui est déjà en mémoire.
- Mais avant cela; nous allons découvrir un serveur HTTP autonome.

Programmation web en Python

Serveur HTTP autonome

Serveur HTTP autonome

■ Le module Python **http.server** contient un serveur HTTP déjà implémenté et prêt à l'emploi

- 2 classes :
 - **HTTPServer** : le serveur proprement dit
 - **BaseHTTPRequestHandler** : l'objet chargé de répondre aux requêtes HTTP
 - Cette classe est faite pour être héritée

Serveur HTTP autonome

■ Mode d'emploi du serveur :

```
#coding:utf-8

import http.server

class RequestHandler(http.server.BaseHTTPRequestHandler):
    def do_GET(self) : # Méthode appelée pour les requêtes GET
        self.client_address # Adresse du client : (hôte, port)
        self.path # URL demandée
        if error :
            self.send_error(404) # Envoie une erreur HTTP (ici n°404 rien trouvé à l'@
spécifié)
            self.send_response(200) # Envoie une réponse HTTP OK
            self.send_header("En-tête1", "Valeur1") # Envoi d'un en-tête HTTP
            self.send_header("En-tête2", "Valeur2") # Envoi d'un en-tête HTTP
            self.end_headers() # Termine les en-têtes
            self.wfile.write(binary_data) # Envoie les données

    def do_POST(self) : # Méthode appelée pour les requêtes POST
        longueur = int(self.headers["content-length"])
        binary_data = self.rfile.read(longueur) # Lit les données envoyées par le client
etc...

# Crée le serveur et le fait tourner indéfiniment
httpd = http.server.HTTPServer(("localhost", 8080), RequestHandler)
httpd.serve_forever() # desservir le serveur de manière continue
```

Serveur HTTP autonome

■ self.path contient l'URL demandé. Il faut d'abord **import urllib.parse**

- l'URL commence par / (= pas de partie protocole ni serveur)
- elle peut contenir une partie « chemin », « requête » (?) et/ou « fragment » (#)
- Exemple :
 - **/index.html**
 - **/page.html?x=1&y=2**
 - **/livre.html#chapitre1** (fragment permet de désigner une partie bien précise du document)

■ Les données au format texte (Unicode, **str** en Python) doivent être encodées en binaire (**bytes**)

- On utilise en général l'encodage UTF-8
 - encodage : **binary_data = texte.encode("utf-8")**
 - décodage : **texte = binary_data.decode("utf-8")**
- Il faut alors spécifier l'encodage dans le type MIME :
 - **text/html => text/html; charset=utf-8**

Serveur HTTP autonome: Exemple 1

```
>>> import http.server

>>> class Hello(http.server.BaseHTTPRequestHandler):

...     def do_GET(self):

...         self.send_response(200)

...         self.send_header("Content-type", "text/html; charset=utf-8")

...         self.end_headers()

...         html = """<!DOCTYPE html>

...             <head>

...                 <title>Da programme</title>

...             </head>

...             <body>

...                 "Hello %s , Vous avez demandé la page %s."%(self.client_address, self.path)

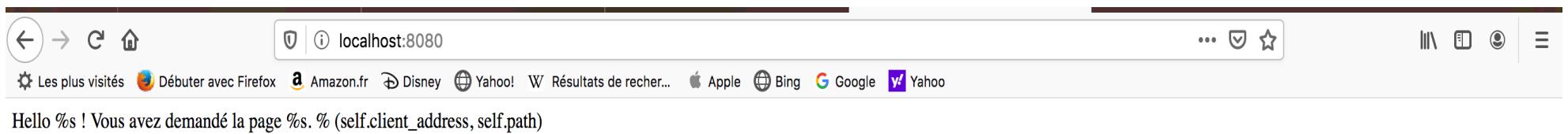
...             </body>

...         </html>"""

...     def do_POST(self):
```

Serveur HTTP autonome: Exemple 1

```
...     html = html.encode("utf-8")
...
self.wfile.write(html)
...
...
>>> httpd = http.server.HTTPServer(("localhost", 8080), Hello)
>>> print("Lancement du serveur sur http://localhost:8080 ...")
Lancement du serveur sur http://localhost:8080 ...
>>> httpd.serve_forever()
```



```
>>> httpd.serve_forever()
127.0.0.1 - - [04/Mar/2020 15:24:56] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Mar/2020 15:24:56] "GET /favicon.ico HTTP/1.1" 200 -
```

Serveur HTTP autonome: Exemple 2

```
>>> import http.server

>>> class Hello(http.server.BaseHTTPRequestHandler):

...     def do_GET(self):

...         self.send_response(200)

...         self.send_header("Content-type", "text/html; charset=utf-8")

...         self.end_headers()

...         html = """<!DOCTYPE html>

...             <head>

...                 <title>Da programme</title>

...             </head>
```

Serveur HTTP autonome: Exemple 2

```
...
<body>
    ...
        <form action="/index.py" method="post">
            ...
                <input type="text" prenom="name" value="Votre prenom" />
            ...
                <input type="submit" name="send" value="Envoyer information au serveur">
            ...
        </form>
    ...
    </body>
...
</html>
...
"""
...
    html = html.encode("utf-8")
...
    self.wfile.write(html)
...
...
>>> httpd = http.server.HTTPServer(("localhost", 8485), Hello)
>>> httpd.serve_forever()
```

Serveur HTTP autonome: Exemple 2

A screenshot of a Firefox browser window. The address bar shows 'localhost:8484'. Below the address bar, there's a search bar with 'Amadou Dahirou' and a button labeled 'Envoyer information au serveur'. The main content area of the browser is empty, showing a white page. At the bottom of the browser window, there's a terminal-like output showing log entries:

```
127.0.0.1 - - [04/Mar/2020 16:24:43] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Mar/2020 16:24:43] "GET /favicon.ico HTTP/1.1" 200 -
127.0.0.1 - - [04/Mar/2020 16:24:57] "GET /index.py?send=Envoyer+information+au+serveur HTTP/1.1" 200 -
```

A noter que python ne fait pas de différences entre POST et GET, vous pouvez passer une variable dans l'url le résultat sera le même:

`http://localhost:8485/index.py?name=amadou`

Serveur HTTP autonome

■ Exercice HTTPD 1 :

- Nous allons programmer un tableau blanc interactif avec deux pages :
 - **lire.html** qui affiche le contenu du tableau et un formulaire avec un champ texte et un bouton d'envoi
 - **écrire.html** qui écrit la ligne saisie et contient un lien pour retourner sur la page précédente

Page <http://localhost:8080/lire.html>

...

Envoyer

...

bla bla bla

Envoyer

Page <http://localhost:8080/lire.html>

...

bla bla bla

Envoyer

Page <http://localhost:8080/écrire.html>

Ligne écrite !
[Retourner lire le tableau](#)

Serveur HTTP autonome

■ Exercice HTTPD 1 :

- Importer les modules `urllib.parse` et `http.server`
- Créer une variable global `TEXTE` qui vaut "..."
- Créer une sous-classe de `http.server.BaseHTTPRequestHandler`
 - Ajouter une méthode `do_GET()` qui :
 - Si l'URL est `/lire.html` : retourne une page HTML contenant :
 - le contenu de la variable `TEXTE`
 - le formulaire avec
 - la méthode POST et « l'action » `ecrire.html`
 - un champ de type texte nommé « ligne »
 - un bouton submit
 - Pour les autres URL, retourne une erreur **404 NOT FOUND**
 - Tester cette page

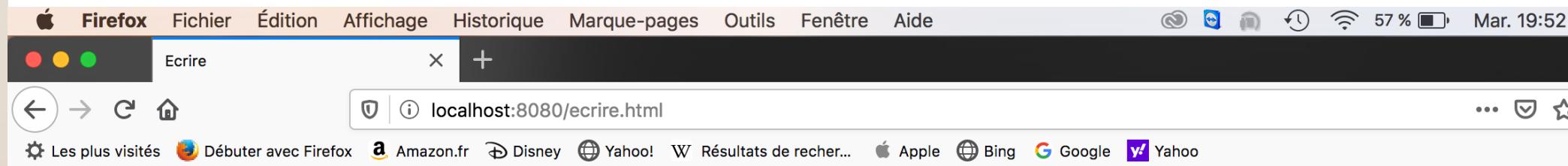
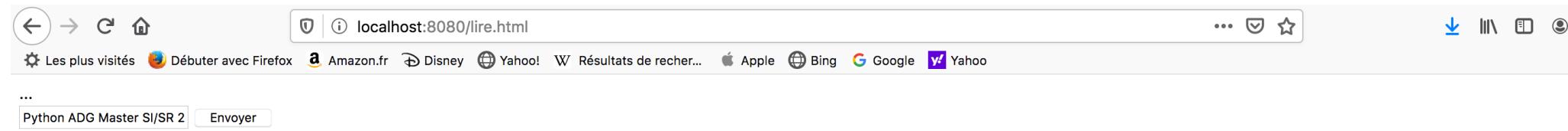
Serveur HTTP autonome

■ Exercice HTTPD 1 (suite) :

- Dans la sous-classe de `http.server.BaseHTTPRequestHandler`
 - Ajouter une méthode `do_POST()` qui :
 - Si l'URL est `/ecrire.html` :
 - récupérer les paramètres du formulaire passés en POST :
 - obtenir la longueur des données à partir des en-têtes
 - lire les données binaires
 - les décoder (UTF-8)
 - utiliser `urllib.parse.parse_qs()` puis `dict()` pour obtenir un dictionnaire des paramètres
 - ajouter la valeur du paramètre «ligne» à la variable global `TEXTE`
 - retourner une page HTML avec un lien vers `lire.html`
 - Retourne une erreur `404 NOT FOUND` pour les autres URL
 - Tester le tableau interactif

Implémentation

result



Ligne écrite !
[Retourner lire le tableau](#)

result

Firefox Fichier Édition Affichage Historique Marque-pages Outils Fenêtre Aide

Lire

localhost:8080/lire.html

Les plus visités Débuter avec Firefox Amazon.fr Disney Yahoo! Résultats de recher... Apple Bing Google Yahoo

...

Python ADG Master SI/SR 2020

Une deuxième saisie Envoyer

Firefox Fichier Édition Affichage Historique Marque-pages Outils Fenêtre Aide

Ecrire

localhost:8080/ecrire.html

Les plus visités Débuter avec Firefox Amazon.fr Disney Yahoo! Résultats de recher... Apple Bing Google Yahoo

Ligne écrite !

[Retourner lire le tableau](#)

result

A screenshot of a Firefox browser window. The title bar says "Lire". The address bar shows "localhost:8080/lire.html". The page content includes a text input field with the value "Fin de la saisie" and a button labeled "Envoyer". The browser interface is in French.

A screenshot of a Firefox browser window, identical to the one above, but with the text input field empty. The browser interface is in French.

Manipuler les URL

■ Le module `urllib.parse` permet de découper les URL :

```
import urllib.parse

url = urllib.parse.urlparse("http://serveur.fr:80/page.html?x=1&y=2#frag")

url.scheme      # = "http"
url.netloc     # = "serveur.fr:80"
url.hostname   # = "serveur.fr"
url.port        # = 80
url.path        # = "page.html"
url.query       # = "x=1&y=2" ——————
url.fragment    # = "frag"

params = dict(urllib.parse.parse_qs(url.query))
print(params) # => { "x" : "1", "y" : "2" }
```

■ Il permet aussi de les assembler :

- ◆ `urllib.parse.urlunparse(["http", "serveur.fr", "page.html", "", "x=1", "frag"])`
=> "http://serveur.fr/page.html?x=1#frag"
- ◆ `urllib.parse.urlencode({"x" : "1", "y" : "2" })`
=> "x=1&y=2"

Manipuler les URL

■ Le module `urllib.parse` permet aussi de gérer les accents et les caractères spéciaux dans les URL :

```
>>> import urllib.parse  
  
>>> urllib.parse.quote("page_accentuée.html")  
"page_accentu%C3%A9e.html"  
  
>>> urllib.parse.unquote("page_accentu%C3%A9e.html")  
"page_accentuée.html"
```

■ Pour récupérer les paramètres des requêtes GET ou POST sous la forme d'un dictionnaire Python :

```
def do_GET(self) :  
    url = urllib.parse.urlparse(self.path)  
    params = dict(urllib.parse.parse_qsl(url.query))  
    ...  
  
def do_POST(self) :  
    binary_data = self.rfile.read(int(self.headers["content-length"]))  
    params = dict(urllib.parse.parse_qsl(binary_data.decode("utf-8")))  
    ...
```

Cache HTTP

■ L'entête HTTP **Cache-Control** permet de définir les options de mise en cache lors de la réponse HTTP

- Nombreuses options
- La plus basique : `max-age=Xsecondes`
 - Exemple `max-age=3600` : mise en cache pendant 1 heure
- À envoyer au client avec la méthode `send_header()` de `http.server.BaseHTTPRequestHandler`
 - Exemple : `self.send_header("Cache-Control", "max-age=3600")`

Programmation web en Python

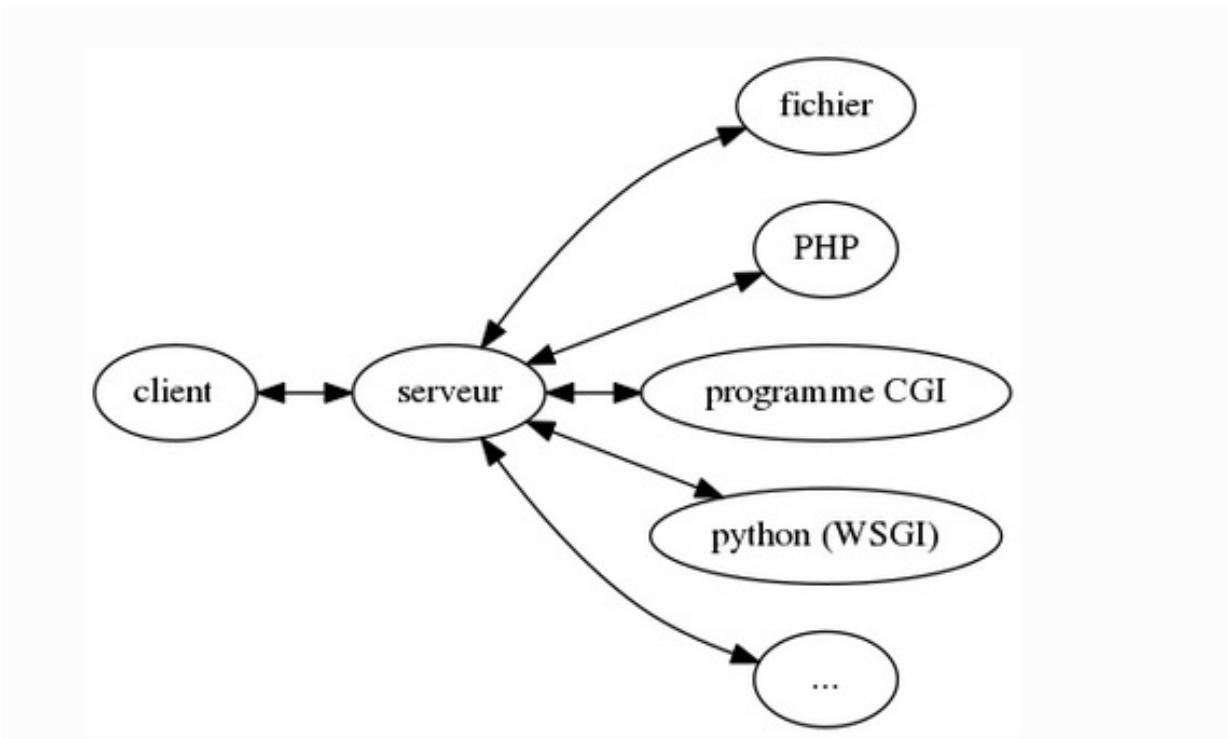
WSGI

WSGI

- WSGI (Web Server Gateway Interface) est l'approche recommandée pour la programmation web en Python
 - ◆ WSGI est une évolution du CGI

Serveur et application

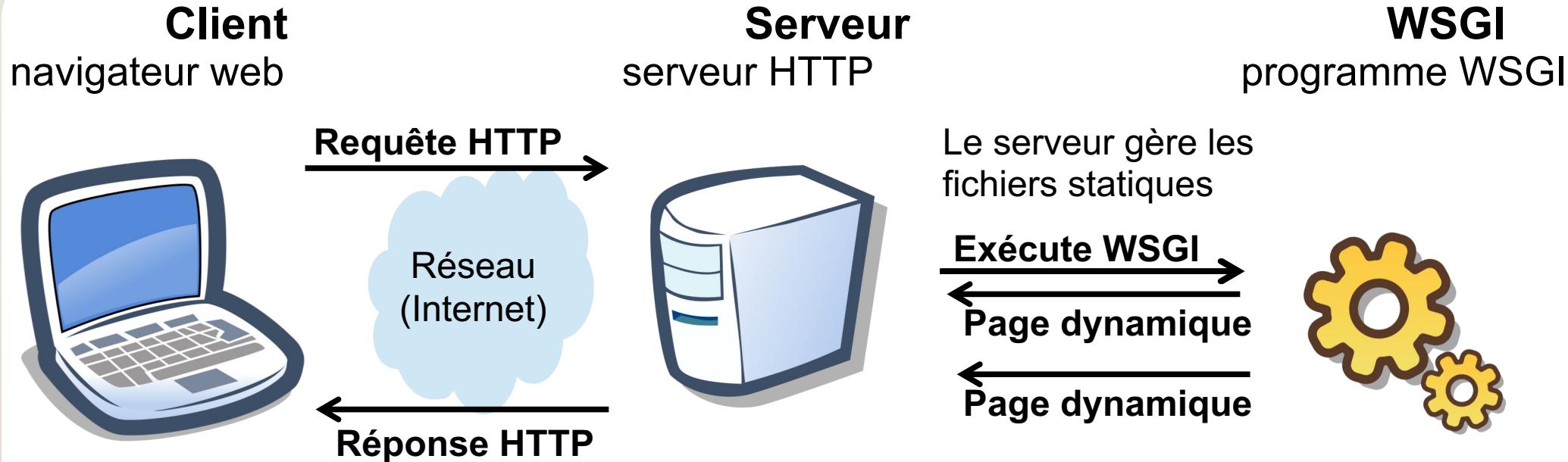
- Les ressources du serveur peuvent être gérées de différentes manières



Serveur et application: Explications

- Certaines ressources sont stockées directement dans des fichiers statiques (dont le nom est en général corrélé avec le chemin de l'URL).
- Certaines ressources sont gérées par un script PHP.
- Certaines ressources sont gérées par un programme, répondant à certains standards, comme
 - CGI,
 - WSGI, que nous allons décrire dans ce cours,
 - ...

WSGI



■ WSGI (Web Server Gateway Interface)

- Le serveur gère les pages et les fichiers statiques
- Le programme WSGI est exécuté au lancement du serveur, et une fonction **application()** est appelée pour chaque page dynamique
 - Un seul chargement du programme WSGI
 - Possibilité de conserver des données entre deux appels

Serveur WSGI

■ Pour le développement :

- **wsgiref**
 - Intégré à Python
 - Mais ne gère pas les fichiers statiques
- **Django**
- **Werkzeug (<http://werkzeug.pocoo.org>)**
 - Un « couteau suisse » pour le développement web en Python
 - Inclut un serveur WSGI minimalist et facile d'emploi
 - Gère les fichiers statiques

■ En production :

- Apache + mod_wsgi (NB ne pas utiliser mod_python !)
- Nginx
- (IIS)

Application WSGI

■ Une application est une fonction de la forme :

```
def application(env, start_response):  
  
    status = "200 OK"  
    headers = [("Content-type", "text/html; charset=utf-8")]  
    start_response(status, headers)  
    html = "<html> ... </html>"  
    return [html.encode("utf-8")]
```

- La fonction est appelée pour chaque page dynamique
 - Le second paramètre **start_response()** est une fonction à appeler pour envoyer l'en-tête HTTP. On génère ici les méta-données de la réponse. Elle prend 2 paramètres :
 - **status** : le code de réponse HTTP
 - **headers** : les en-têtes HTTP, sous la forme d'une liste de tuple (nom d'en-tête, valeur)
 - La fonction doit retourner une liste contenant le document à retourner au client; ie une liste sous forme d'octets

Encodage des documents

■ Le document retourné doit être encodé en binaire

- Soit un fichier texte encodé
 - Le document au format texte (Unicode, **str** en Python) doit être encodé en binaire (**bytes**)
 - On utilise en général l'encodage Unicode UTF-8
 - ◆ encodage : **binary_data = texte.encode("utf8")**
 - ◆ décodage : **texte = binary_data.decode("utf8")**
 - Il faut alors spécifier l'encodage dans le type MIME (en-tête Content-Type) :
 - ◆ **text/html => text/html; charset=utf-8**
- Soit un fichier binaire (ex image PNG ou JPEG) :

```
jpeg = open("image.jpeg", "rb").read()
status = "200 OK"
headers = [("Content-type", "image/jpeg")]
start_response(status, headers)
return [jpeg]
```

Application WSGI

- Le premier paramètre **env** (pour environnement) est un dictionnaire qui contient notamment toute l'information décrivant la requête. Il contient divers paramètres :
 - **env["PATH_INFO"]** : le chemin demandé par le client (ex `/dir/page.html`)
 - **env["REQUEST_METHOD"]** : la commande HTTP (ex **GET**, **POST**)
 - **env["QUERY_STRING"]** : les paramètres passés dans l'URL après le « ? » (ex `x=1&y=2`)
 - **env["SERVER_NAME"]** : nom du serveur où tourne le script WSGI
 - **env["SERVER_PORT"]** : port du serveur où tourne le script WSGI
 - **env["wsgi.errors"]** : un (pseudo-)fichier ouvert en écriture où l'on peut écrire des messages d'erreur
- Pour les requêtes **POST** et **PUT** :
 - **env["CONTENT_LENGTH"]** : longueur du document envoyé
 - **env["wsgi.input"]** : un (pseudo-)fichier ouvert en lecture où l'on peut lire le document envoyé

Paramètres de la requête HTTP

■ HTTP permet de passer des paramètres de plusieurs manières

- Dans l'URL, après le « ? »

- Ex : `http://mon_serveur/ma_page.html?x=1&y=2`

■ Pour les requêtes de type POST

- Dans le document envoyé avec la requête HTTP

- Deux possibilités d'encodage, le choix se fait dans le formulaire HTML :

```
<form method="POST" enctype="multipart/form-data"  
      action="page_reponse.html">...</form>
```

- **application/x-www-form-urlencoded**

- Encodé de la même manière que les paramètres passés dans l'URL

- Ex : `x=1&y=2`

- **multipart/form-data**

- Encodé sous la forme d'un message multipartie

- Obligatoire pour l'envoi de fichier

Paramètres de la requête HTTP

- **multipart/form-data** : un document par paramètre

- Exemple :

-----1627528211999312422155874322

Content-Disposition: form-data; name="titre"

La Liberté guidant le peuple

-----1627528211999312422155874322

Content-Disposition: form-data; name="année"

1830

-----1627528211999312422155874322

Content-Disposition: form-data; name="photo";
filename="plan.png"

Content-Type: image/png

_PNG poejfoz »'jfopéfjp »zo'fjo »zmaifj (donnée binaire PNG)

Paramètres de la requête HTTP

● multipart/form-data : Exemple 2

- <form action="http://localhost:8000" method="post" enctype="multipart/form-data">
- <p><input type="text" name="text1" value="text default">
- <p><input type="text" name="text2" value="aωb">
- <p><input type="file" name="file1">
- <p><input type="file" name="file2">
- <p><input type="file" name="file3">
- <p><button type="submit">Submit</button>
- </form>

WSGI : récupération des paramètres de la requête HTTP

■ Pour les paramètres passés dans l'URL :

- ◆ On utilise le module `urllib.parse` et la fonction `parse_qs()`, qui retourne un dictionnaire :

```
import urllib.parse

def application(env, start_response):
    query = env["QUERY_STRING"]
    params = dict(urllib.parse.parse_qs(query))
```

WSGI : récupération des paramètres de la requête HTTP

■ Pour les requêtes de type POST :

- Le module **werkzeug** propose une fonction qui gère automatiquement les deux encodages :

```
import werkzeug.formparser
```

```
def application(env, start_response) :
```

```
    stream, params, files = werkzeug.formparser.parse_form_data(env, cls =  
dict)
```

- Retourne 2 valeurs utiles :

- **params** : un dictionnaire avec les paramètres envoyés en POST

- **files** : un dictionnaire avec les fichiers envoyés en POST

```
    params["titre"] # => La Liberté guidant le peuple
```

```
    files["photo"].filename # => plan.png : nom du fichier envoyé  
    files["photo"].save(nom de fichier) # Sauvegarde le fichier
```

- Il est possible de combiner 2 dictionnaires : **dico1.update(dico2)**

Manipuler les URL

■ Le module `urllib.parse` permet de découper les URL :

```
import urllib.parse  
  
url =  
    urllib.parse.urlparse("http://serveur.fr:80/page.html?x=1&y=2#frag")  
  
url.scheme      # = "http"  
url.netloc       # = "serveur.fr:80"  
url.hostname     # = "serveur.fr"  
url.port         # = 80  
url.path          # = "page.html"  
url.query         # = "x=1&y=2"  
url.fragment      # = "frag"  
  
params = dict(urllib.parse.parse_qs(url.query))  
print(params) # => { "x" : "1", "y" : "2" }
```

Manipuler les URL

Il permet aussi de les assembler :

- ◆ `urllib.parse.urlunparse(["http", "serveur.fr", "page.html", "", "x=1", "frag"])`
=> "http://serveur.fr/page.html?x=1#frag"
- ◆ `urllib.parse.urlencode({"x" : "1", "y" : "2" })`
=> "x=1&y=2"

WSGI : envoi d'erreur HTTP

■ Pour générer une erreur HTTP, on utilise le statut correspondant

◆ Exemple pour une erreur 404 :

```
def application(env, start_response):  
    status = "404 NOT FOUND"  
    headers = [("Content-type", "text/plain")]  
    start_response(status, headers)  
    text = "Not found."  
    return [text.encode("utf-8")]
```

Lancement du serveur WSGI

■ Pour lancer le serveur Werkzeug en Python :

```
import werkzeug.serving  
  
def application(env, start_response) :  
    ...  
  
werkzeug.serving.run_simple("adresse", port, application,  
    static_files = {"URL statique" : "fichier ou répertoire local",...})
```

◆ **static_files** est un dictionnaire qui fait correspondre les chemins statiques à des fichiers ou des répertoires locaux

Mon premier WSGI

■ Exercice WSGI 1 :

- ◆ Nous allons programmer un tableau interactif avec deux pages :
 - **lire.html** qui affiche le contenu du tableau et un formulaire avec un champ texte et un bouton d'envoi
 - **écrire.html** qui écrit la ligne saisie et contient un lien pour retourner sur la page précédente

Page <http://localhost:8080/lire.html>

...

Envoyer

...

bla bla bla

Envoyer

Page <http://localhost:8080/lire.html>

...

bla bla bla

Envoyer

Page <http://localhost:8080/écrire.html>

Ligne écrite !
[Retourner lire le tableau](#)

Mon premier WSGI

■ Exercice WSGI 1 :

- Importer les modules `urllib.parse` et `werkzeug.serving`
- Créer une variable global `TEXTE` qui vaut "..."
- Créer une fonction `application()` :
 - Si le chemin demandé est `/lire.html` : retourne une page HTML contenant :
 - le contenu de la variable `TEXTE`
 - un formulaire avec
 - la méthode GET et « l'action » `ecrire.html`
 - un champ de type texte nommé « ligne »
 - un bouton submit
 - Pour les autres URL, retourne une erreur **404 NOT FOUND**
- Tester cette page

Mon premier WSGI

■ Exercice WSGI 1 (Suite) :

- ◆ Dans la fonction **application()** :
 - Ajouter la prise en charge du chemin **/ecrire.html** :
 - récupérer les paramètres du formulaire passés en GET :
 - utiliser **urllib.parse.parse_qs()** puis **dict()** pour obtenir un dictionnaire des paramètres
 - ajouter la valeur du paramètre « ligne » à la variable global **TEXTE**
 - retourner une page HTML avec un lien vers **lire.html**
 - ◆ Tester le tableau interactif

Mon premier WSGI

Implémentation

Serveur web WSGI

- **Werkzeug**: bibliothèque d'utilitaires WSGI pour Python.
<https://werkzeug.palletsprojects.com/en/2.0.x/>
- **Framework Django**: cadre de développement web open source en Python. Il a pour but de rendre le développement web 2.0 simple et rapide. Django prend actuellement en charge deux interfaces : WSGI et ASGI.
<https://www.djangoproject.com/>
- **Flask** : micro framework open-source de développement web en Python qui se base sur deux modules werkzeug et jinja2. *<https://flask.palletsprojects.com/en/2.0.x/>*

Déploiement

■ De nombreux hébergeurs en ligne proposent l'hébergement de site en Python :

- ◆ <https://www.gandi.net/hebergement/simple?language=python>
- ◆ <https://www.pythonanywhere.com/>
- ◆ <https://www.webfaction.com/>
- ◆ <https://www.openshift.com/>
- ◆ ...

■ Une application WSGI + un répertoire de fichiers statiques

Programmation web en Python

TP avec Django

Frameworks

■ Toute application Web se compose d'une **partie backend** ou côté serveur et d'une **partie front-end**. Par conséquent, il existe des **frameworks** de développement web **front-end** et **back-end**.

■ Framework : Ensemble d'outils environnement qui nous permettent de faire le développement d'un site web très facilement

■ Exemples de framework:

- Sous php: Symphony, Laravel, etc.
- Sous Ruby: Ruby on Rails, Roda, Camping
- Sous python: Django (open source), Flask (licence BSD), Bottle (licence MIT), CherryPy (open source)

■ **Django** est un framework web python open-source consacré au développement web 2.0

■ Une grande communauté de développeurs

■ Il est donc clairement orienté pour les développeurs ayant comme besoin de produire un projet solide rapidement et sans surprise ... c'est à dire à tous les développeurs !

■ Django s'inspire du modèle MVC (disons plutôt ici **MVT**), c'est-à-dire que la structure du framework sépare les données (models) qui sont séparées des traitements (controller) qui sont eux-mêmes séparés de la vue (view / template).

■ Sous Django: **Modèle** gère la base de données - le **View** joue le rôle de d'orchestration ou contrôleur - le **Template** gère l'affichage et joue ici e rôle de vue

Présentation Django

- **Django est un framework web python open-source consacré au développement web 2.0**
- **Il est donc clairement orienté pour les développeurs ayant comme besoin de produire un projet solide rapidement et sans surprise ... c'est à dire à tous les développeurs !**
- **Django s'inspire du modèle MVC (disons plutot MVT), c'est-à-dire que la structure du framework sépare les données (models) qui sont séparées des traitements (controller) qui sont eux-mêmes séparés de la vue (view / template).**
- **Django: Idéal pour un projet collaboratif**
- **Django est apprécié également des grandes entreprises telles que Pinterest, Instagram, Libération, 20 minutes, Mozilla, etc.**

Pourquoi le choix de Django

- simplicité d'apprentissage
- efficacité de votre développement
- solidité de vos projets
- sécurité finale
- facilité de maintenance
- facilité d'intégration de nouveaux développeurs
- projets annexes comme DRF

Les différentes étapes dans Django

- Crédit d'un environnement virtuel dédié à notre projet. Ce sera en quelque sorte une boîte où on mettra seulement les paquets dont on aura besoin.
- Activation de l'environnement virtuel
- Installation de Django et ses dépendances
- Crédit d'un projet Django
- Démarrage de l'application et lancement du site
- Crédit d'URL, de templates et affichage de page web selon l'URL
- Crédit d'applications
- Passage de variables dans une page web

Création d'un environnement virtuel dédié au projet

- L'environnement virtuel symbolise une boîte où on mettra nos paquets
- Crédit de l'environnement virtuel « **gueye** »
 - Créer un dossier « Django » dans bureau. Ce dossier va contenir notre projet django
 - S'y déplacer et taper la commande: *python3 –m venv gueye*
- activation de l'environnement virtuel

```
[amadoudahirougueye@MBP-de-Amadou Django % source gueye/bin/activate  
(gueye) amadoudahirougueye@MBP-de-Amadou Django % ]
```

- Pour désactiver l'environnement virtuel, tapez:

deactivate gueye/bin/activate

- Activation sous windows

./gueye/script/activate.bat ou ./gueye/Scripts/activate

Installation de Django

A partir de là, on peut installer Django avec ses dépendances

pip install django

```
[(gueye) amadoudahirougueye@MBP-de-Amadou Django % pip install django
Collecting django
  Downloading Django-4.0.6-py3-none-any.whl (8.0 MB)
    ━━━━━━━━━━━━━━━━━━━━ 8.0/8.0 MB 79.8 kB/s eta 0:00:00
Collecting asgiref<4,>=3.4.1
  Downloading asgiref-3.5.2-py3-none-any.whl (22 kB)
Collecting sqlparse>=0.2.2
  Downloading sqlparse-0.4.2-py3-none-any.whl (42 kB)
    ━━━━━━━━━━━━━━━━ 42.3/42.3 KB 741.0 kB/s eta 0:00:00
Installing collected packages: sqlparse, asgiref, django
Successfully installed asgiref-3.5.2 django-4.0.6 sqlparse-0.4.2
WARNING: You are using pip version 22.0.4; however, version 22.1.2 is available.
You should consider upgrading via the '/Users/amadoudahirougueye/Desktop/Django/gueye/bin/python3 -m pip install --upgrade pip' command.
(gueye) amadoudahirougueye@MBP-de-Amadou Django % ]
```

■ Vérification:

```
[(gueye) amadoudahirougueye@MBP-de-Amadou Django % python3
Python 3.10.5 (v3.10.5:f377153967, Jun  6 2022, 12:36:10) [Clang 13.0.0 (clang-1
300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
[>>> import django
>>> ]
```

■ Pour avoir la version

print(django.__version__)
4.0.6

■ **NB:** Pour forcer une version de Django: ***pip install django==3.10.3***

Création de projet sous Django

■ Crédit à Amadou pour la création du projet

django-admin startproject da

```
[(gueye) amadoudahirougueye@MBP-de-Amadou Django % django-admin startproject da
[(gueye) amadoudahirougueye@MBP-de-Amadou Django % ls
da      gueye
[(gueye) amadoudahirougueye@MBP-de-Amadou Django % cd da/
[(gueye) amadoudahirougueye@MBP-de-Amadou da % code .
```

The screenshot shows a dark-themed interface of VS Code. On the left, the Explorer sidebar displays a project structure with a folder named 'DA' containing a subfolder 'da' which contains files: __init__.py, asgi.py, settings.py, urls.py, wsgi.py, and manage.py. The 'manage.py' file is currently open in the main editor area. The code in 'manage.py' is as follows:

```
#!/usr/bin/env python
"""
Django's command-line utility for administrative tasks.
"""

import os
import sys

def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'da.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)

if __name__ == '__main__':
    main()
```

A small modal window at the bottom center asks if the user wants to install recommended Python extensions, with buttons for 'Installer' and 'Afficher les recommandations'.

NB: Pour lancer vs code en ligne de commande:

- Launch VS Code.
- Open the Command Palette (Cmd+Shift+P) and type 'shell command' to find the Shell Command:
Install 'code' command in PATH

Fichiers du projet da

- **manage.py**: fichier qui nous permet d'exécuter les commandes Django comme pour créer des applications ou pour démarrer le serveur
- **urls.py**: va contenir les urls de django autrement dit les sites qu'on va créer, i.e les pages contact, les blogs et autres.
- **settings.py**: le fichier de configuration qui renferme tout (les templates, la bases de données, le mot de passe, la langue)
- **wsgi.py**: pour gérer les pages web et pour démarrer le serveur web

Démarrage du projet sur le site

■ Pour démarrer le projet da

./manage.py runserver ou python manage.py runserver

```
[gueye] amadoudahirogueye@MBP-de-Amadou da % ./manage.py runserver  
Watching for file changes with StatReloader  
Performing system checks...
```

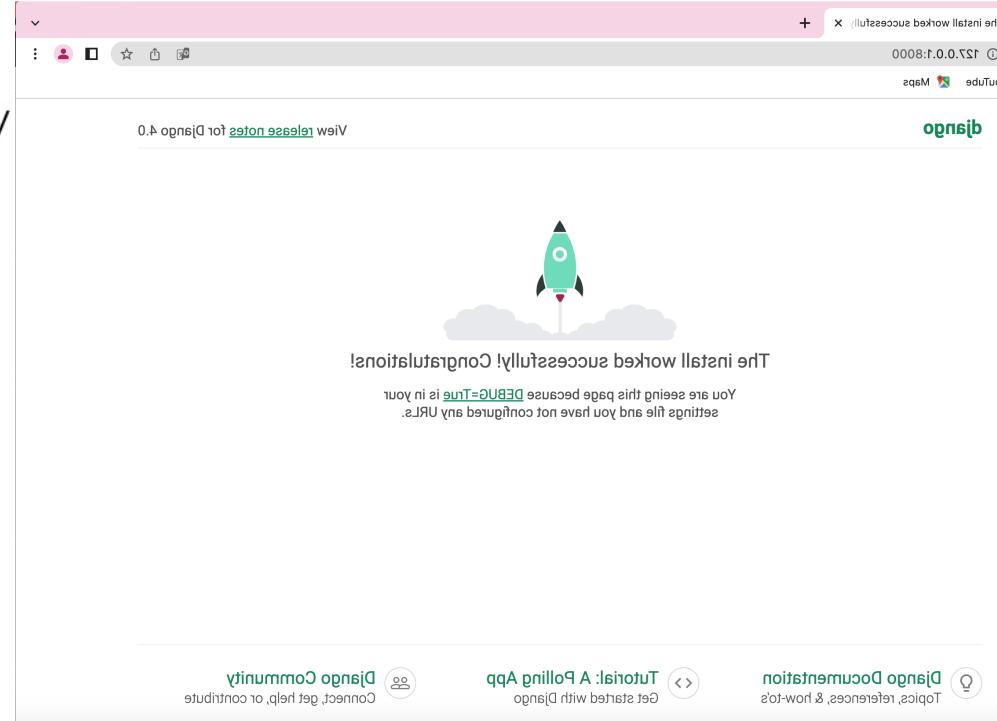
System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.

```
July 13, 2022 - 22:13:54  
Django version 4.0.6, using settings 'da.settings'  
Starting development server at http://127.0.0.1:8000/  
Quit the server with CONTROL-C.
```

■ Clic sur l'adresse pour afficher le site

■ Pour changer la langue aller dans settings



Création d'URL

■ Le fichier **Urls.py** nous gère les urls de notre site. (s'y rendre)

■ Accès à la page d'accueil en tapant sur le site **127.0.0.1:8000/**

■ Préciser python sur l'environnement virtuel de Django à partir de vscode

■ Choisir Enter interpreter path puis la version de python installé, puis aller jusqu'au bin puis p

■ Aller dans urls.py, et mettre

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home), # à ajouter #si on met rien on doit
#accéder à la page d'accueil
]
```

■ Puis importer dans l'url le views: **from . import views**

■ Définir la fonction qui nous permet d'afficher **views.home**. Cette fonction est à définir dans le fichier **views.py** qui gère l'affichage des vues

Création d'URL

- Crédit à la création du fichier views.py et définition de la fonction home qui nous permet d'afficher la page

```
from django.http import HttpResponse
```

```
def home(request):
```

```
    return HttpResponse('<h2>Home page Dahirou</h2>')
```

- Lancer le site sur l'adresse 127.0.0.1:8000/



- Dans ce qui suit, au lieu de retourner des httpResponse, on va retourner des pages web

Création de templates

■ Crédit de templates pour mettre les codes HTML et on appellera ces templates dans la view (contrôleur)

■ Créer dans « da » un dossier **templates** et un fichier **home.html** dans templates

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
</head>
<body>
<h1>Home page Dahirou</h1>
</body>
</html>
```

■ Sur le fichier **views.py**, remplacez la fonction **HttpResponse** par la fonction **render**

```
from django.http import HttpResponse
from django.shortcuts import render

def home(request):
    return render(request, 'home.html')
```

■ Puis actualiser la page web

■ NB: les templates seront recherchés dans le dossier templates

■ Pour cela, aller dans settings et mettez: '**DIRS': ['templates'],**

Exemple de création d'une 2ème URL about

■ Modifications urls.py

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home),
    path('about/', views.about),
```

■ Modifications views.py

```
from django.http import HttpResponseRedirect
from django.shortcuts import render

def home(request):
    return render(request, 'home.html')

def about(request):
    return render(request, 'about.html') #la fonction render nous permet de retourner une page htm
```

■ Puis allez dans le dossier templates et créer about.html

■ Lancer le site sur l'url 127.0.0.1:8000/about/



Création d'une application dédié à Django sur le projet

- Django nous permet de détailler les codes
- A travers notre site, on peut créer une application pour le blog ou pour le forum ou une application gère par exemple les ventes ou, etc.
- Exemple de TP de création d'une application de vente de produits: on va créer des produits, les exposer et les vendre.

```
[(gueye) amadoudahirougueye@MBP-de-Amadou da % django-admin startapp vente  
 (gueye) amadoudahirougueye@MBP-de-Amadou da % ]
```

- Pour que Django le reconnait comme une application, il faut aller dans **settings** et le préciser là-bas dans **apps**

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'vente'  
]
```

- Si on crée des applications, on doit inclure les urls des applications dans l'url du site principal
 - Modifier le fichier urls.py et ajouter la dernière ligne

```
    urlpatterns = [  
        path('admin/', admin.site.urls),  
        path('', views.home),  
        path('about/', views.about),  
        path('produits/', include('vente.urls')),  
    ]
```

- Explication: si on tape **vente/**, on doit rediriger tous les urls au niveau de **vente.urls** créé dans l'application **vente**. Chaque application doit avoir ses propres urls. Créer donc urls.py dans **vente**

Création d'une application au niveau de Django

- Aller dans urls.py de da, faites copie-coller dans urls.py de vente et modifier en:

```
from django.contrib import admin
from django.urls import include, path
from . import views

urlpatterns = [
    path('', views.index),
]
```

- Créer dans vente un dossier templates puis un sous dossier vente, puis un fichier index.html dans le sous dossier vente

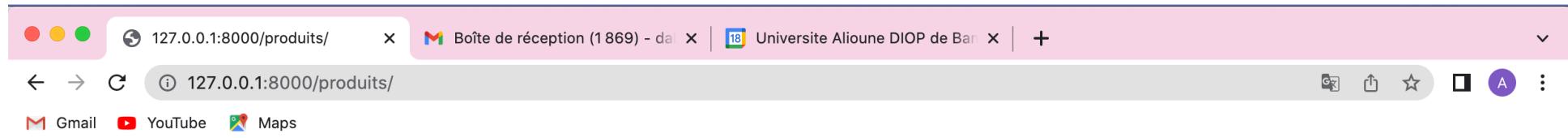
- Modifier le contrôleur views.py

```
from django.shortcuts import render

# Create your views here.
def index(request):
    return render(request, 'vente/index.html')
```

Création d'une application au niveau de Django

- Dans le fichier index.html, mettez là-bas ce que vous voulez visualiser par exemple « vente produits »
- Puis lancer le site et tapez: 127.0.0.1/vente/



Vente produits

TP

- **Créer une application « commandes » pour gérer les commandes des produits**

Passage de variables dans une page web

- Maintenant on veut transférer des variables au niveau de la page index.html.
- Les variables peuvent êtres des chaines de caractères, des listes, des tableaux et autres

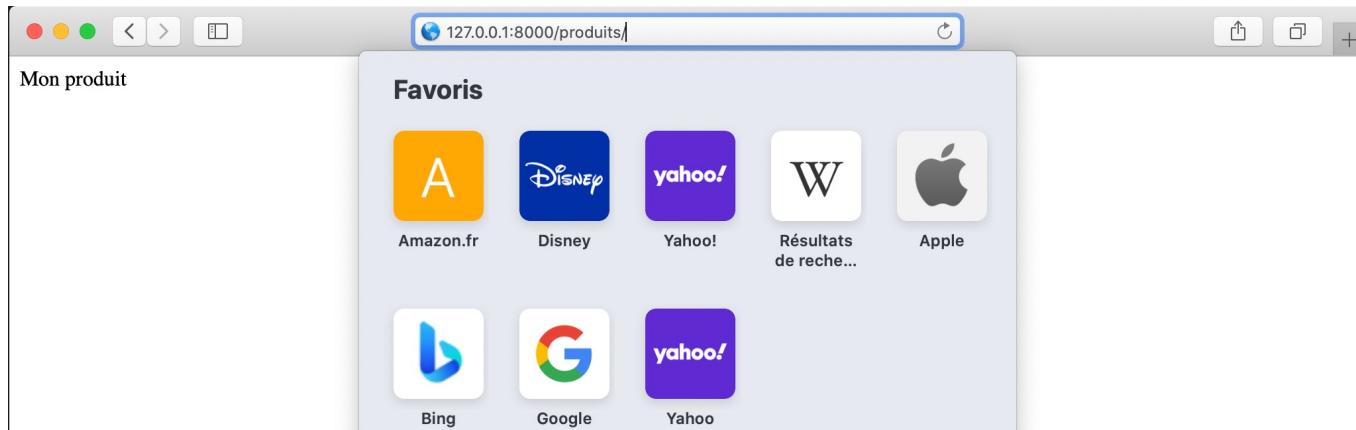
Passage de variables chaines

■ Pour cela, il faut modifier le fichier `views.py` de l'application `vente` et ajouter dans la fonction `index`

- La variable
- Et le contexte (dans le `return`). Le contexte est un couple de clé valeur. La valeur sera la variable

```
vente > views.py > index
1  from django.shortcuts import render
2
3  # Create your views here.
4
5
6  def index(request):
7      produit = 'Mon produit'
8      return render(request, 'vente/index.html', {'produit': produit})
```

■ Au niveau de `index.html` de `vente` , il faut préciser la variable disponible pour avoir accès à `produit` }



Passage de variables listes

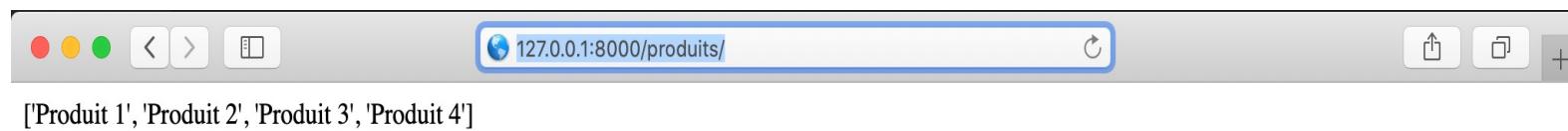
■ Pour cela, il faut modifier le fichier `views.py` de l'application `vente` et ajouter dans la fonction `index`

- La variable
- Et le contexte (dans le `return`). Le contexte est un couple de clé valeur. La valeur sera la variable

```
vente > ⌂ views.py > ⌂ index
1   from django.shortcuts import render
2
3
4   # Create your views here.
5   def index(request):
6       produits=['Produit 1', 'Produit 2', 'Produit 3', 'Produit 4']
7       return render(request, 'vente/index.html', {'produits':produits})
```

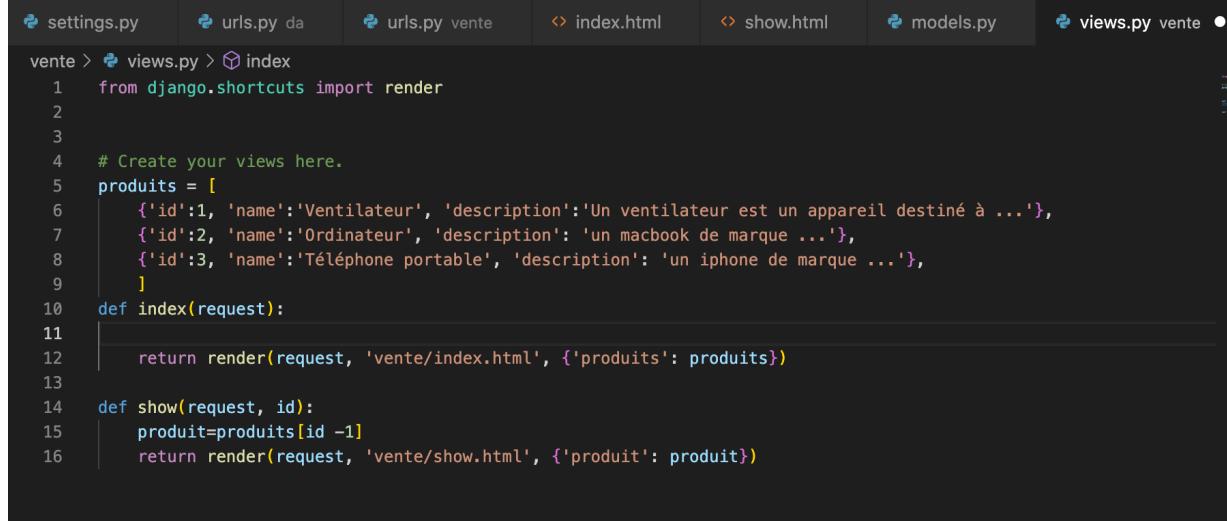
■ Au niveau de `index.html` de `vente` , il faut préciser la variable disponible pour avoir accès

`{{ produit }}`



Passage de variables d'une liste de dictionnaire et affichage d'un élément de la liste

■ Pour cela, il faut modifier le fichier views.py de l'application vente

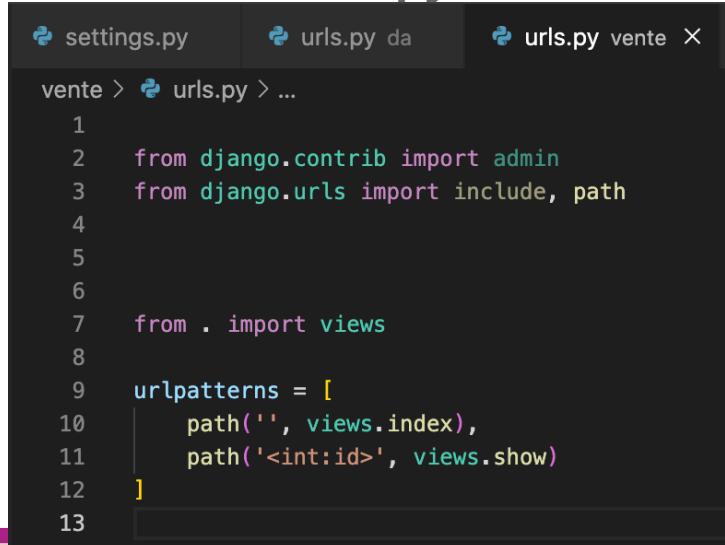


The screenshot shows a code editor with several tabs at the top: settings.py, urls.py da, urls.py vente (which is the active tab), index.html, show.html, models.py, and views.py vente. The views.py file contains Python code for a Django application. It defines two functions: index and show. The index function returns a rendered template 'index.html' with a context variable 'produits' containing a list of products. The show function returns a rendered template 'show.html' with a context variable 'produit' containing a single product from the list.

```
settings.py urls.py da urls.py vente index.html show.html models.py views.py vente

vente > views.py > index
1  from django.shortcuts import render
2
3
4  # Create your views here.
5  produits = [
6      {'id':1, 'name':'Ventilateur', 'description':'Un ventilateur est un appareil destiné à ...'},
7      {'id':2, 'name':'Ordinateur', 'description': 'un macbook de marque ...'},
8      {'id':3, 'name':'Téléphone portable', 'description': 'un iphone de marque ...'},
9  ]
10 def index(request):
11
12     return render(request, 'vente/index.html', {'produits': produits})
13
14 def show(request, id):
15     produit=produits[id -1]
16     return render(request, 'vente/show.html', {'produit': produit})
```

■ Au niveau de urls.py de vente



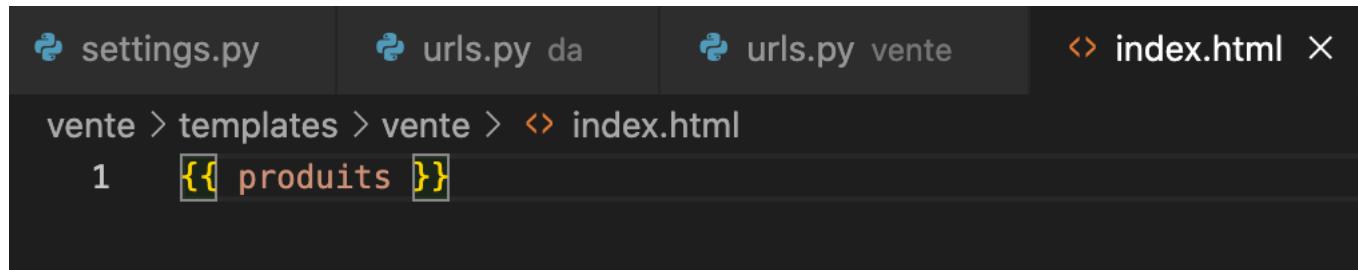
The screenshot shows a code editor with several tabs at the top: settings.py, urls.py da, and urls.py vente (which is the active tab). The urls.py file contains Python code for a Django application. It imports admin and urls from django.contrib and django.urls respectively. It includes a path for the admin interface and a path for the views module's index and show functions.

```
settings.py urls.py da urls.py vente

vente > urls.py > ...
1
2  from django.contrib import admin
3  from django.urls import include, path
4
5
6
7  from . import views
8
9  urlpatterns = [
10     path('', views.index),
11     path('<int:id>', views.show)
12 ]
13
```

Passage de variables d'une liste de dictionnaire et affichage d'un élément de la liste

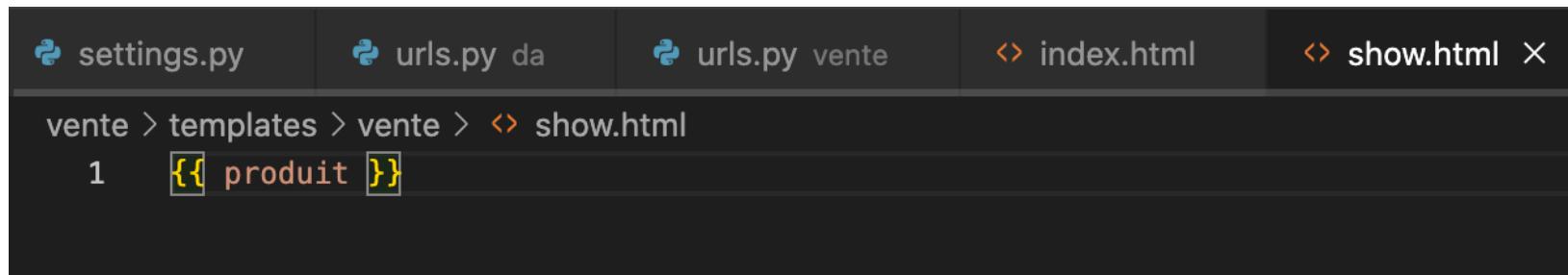
■ Au niveau de index.html



A screenshot of a code editor showing the file `index.html`. The file path is `vente > templates > vente > index.html`. The content of the file is:

```
1  {{ produits }}
```

■ Au niveau de show.html

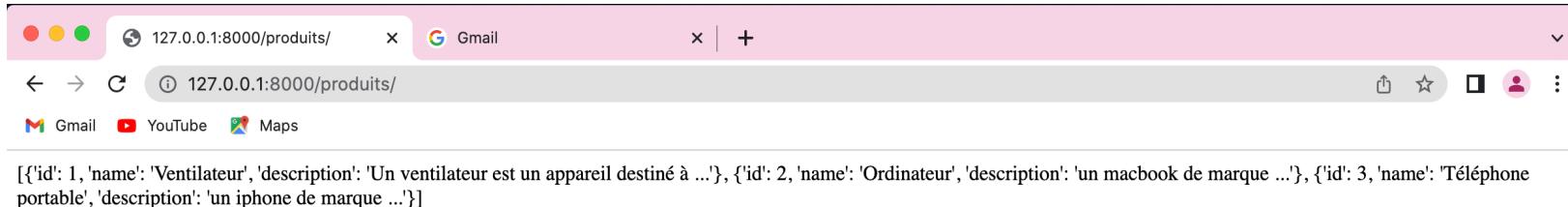


A screenshot of a code editor showing the file `show.html`. The file path is `vente > templates > vente > show.html`. The content of the file is:

```
1  {{ produit }}
```

Passage de variables d'une liste de dictionnaire et affichage d'un élément de la liste

■ si on tape sur l'url produits



```
[{"id": 1, "name": "Ventilateur", "description": "Un ventilateur est un appareil destiné à ..."}, {"id": 2, "name": "Ordinateur", "description": "un macbook de marque ..."}, {"id": 3, "name": "Téléphone portable", "description": "un iphone de marque ..."}]
```

■ si on tape sur l'url produit/1

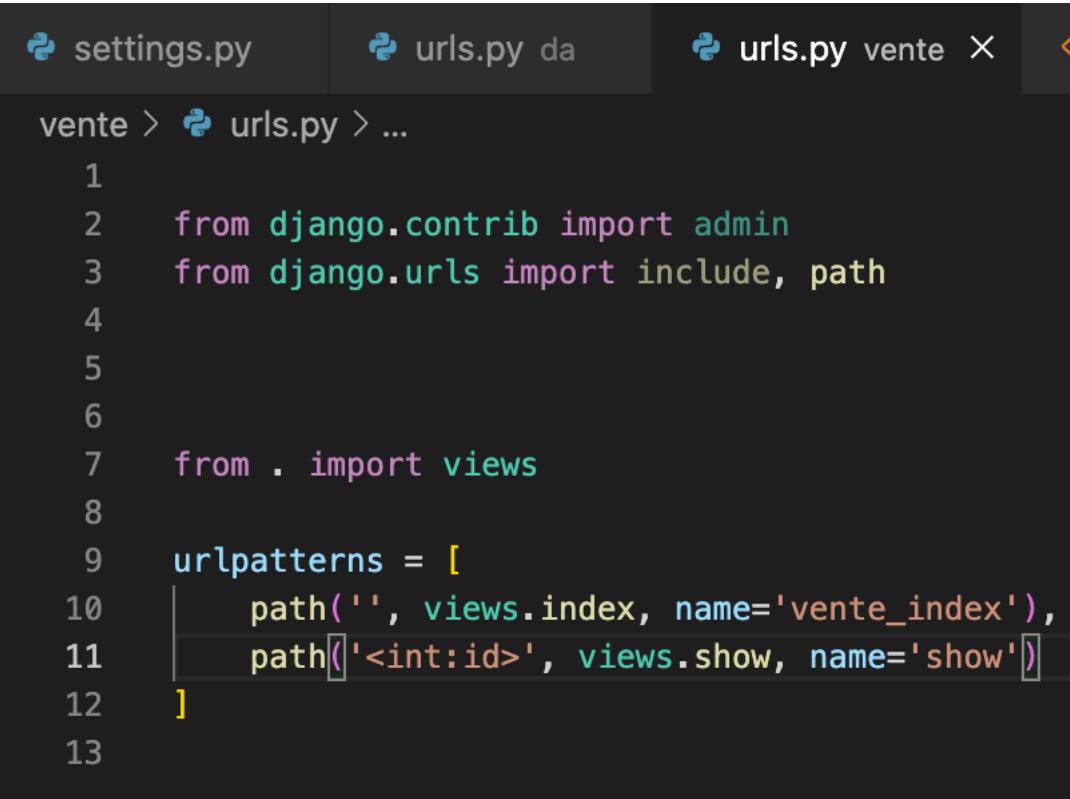


```
{"id": 1, "name": "Ventilateur", "description": "Un ventilateur est un appareil destiné à ..."}
```

■ si on tape sur n'importe quel id on doit avoir le résultat attendu

Name sur les URIs

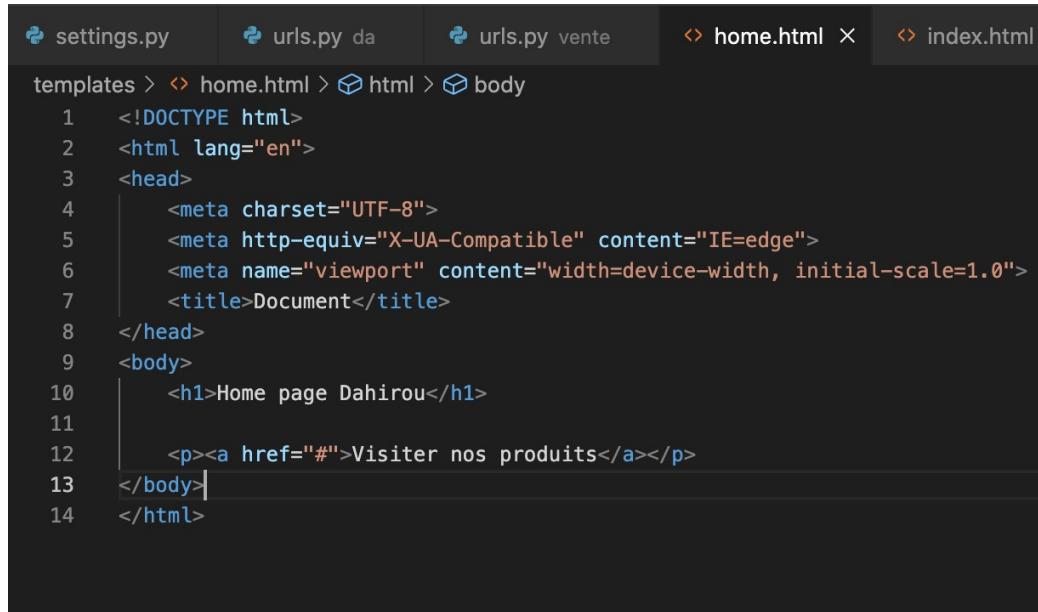
- Ici on va donner un nom à chaque url
- Par exemple pour la page d'accueil de vente, on peut donner comme nom « index »
- i.e que l'URL /produits a comme nom vente_index et l'URL /produits/1 a comme nom sh



```
settings.py urls.py da urls.py vente < ...  
vente > urls.py > ...  
1  
2     from django.contrib import admin  
3     from django.urls import include, path  
4  
5  
6  
7     from . import views  
8  
9     urlpatterns = [  
10         path('', views.index, name='vente_index'),  
11         path('<int:id>', views.show, name='show')  
12     ]  
13
```

Comment naviguer dans des Urls grâce aux liens

Rendons nous d'abord à la page d'accueil home.html



```
settings.py urls.py da urls.py vente home.html index.html

templates > home.html > html > body
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     <h1>Home page Dahirou</h1>
11
12     <p><a href="#">Visiter nos produits</a></p>
13 </body>
14 </html>
```

Tapons la page d'accueil

← → C ⓘ 127.0.0.1:8000

Gmail YouTube Maps

Home page Dahirou

[Visiter nos produits](#)

Utilisation des noms dans les liens href

■ Ici on passe le nom de l'URL

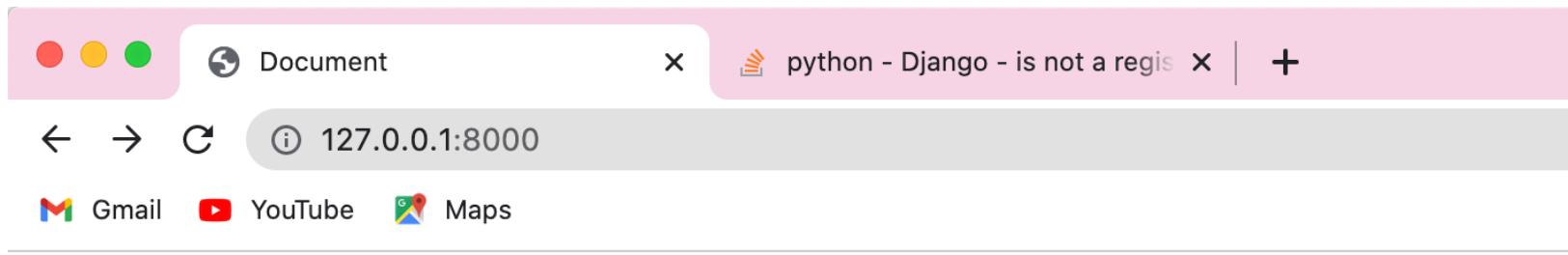
■ Dès fois on peut être amené à donner le nom de l'application suivi de : puis le nom de variable

The screenshot shows a code editor with several tabs at the top: 'settings.py', 'urls.py da', 'urls.py vente', 'home.html' (which is the active tab, indicated by a black dot), 'about.html', and 'index.html'. The 'home.html' tab contains the following code:

```
templates > <> home.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>Document</title>
8 </head>
9 <body>
10 |<h1>Home page Dahirou</h1>
11 |<p><a href="produits/">Visiter nos produits</a></p>
12 |<p><a href="{% url 'vente_index' %}">Visiter nos produits (Avec Name)</a></p>[ [ ] ]
13 </body>
14 </html>
```

Utilisation des name dans les liens href

■ Si on actualise la page, ça donne:



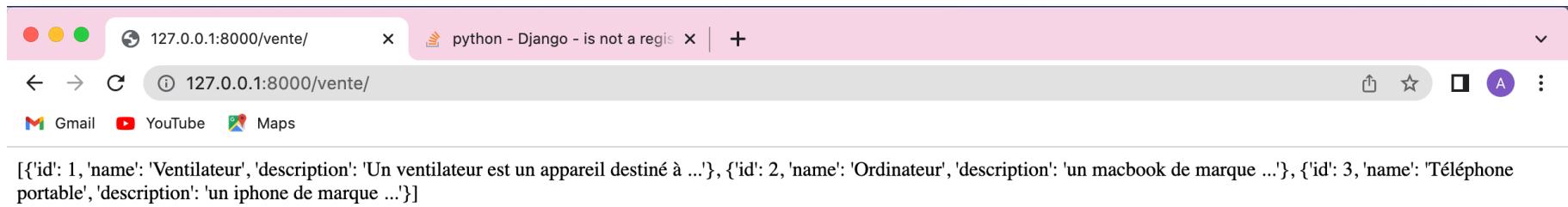
Home page Dahirou

[Visiter nos produits](#)

[Visiter nos produits \(Avec Name\)](#)

Utilisation des name dans les liens href

■ Un clic sur le deuxième lien donne



Comment naviguer dans des URLs grâce aux liens

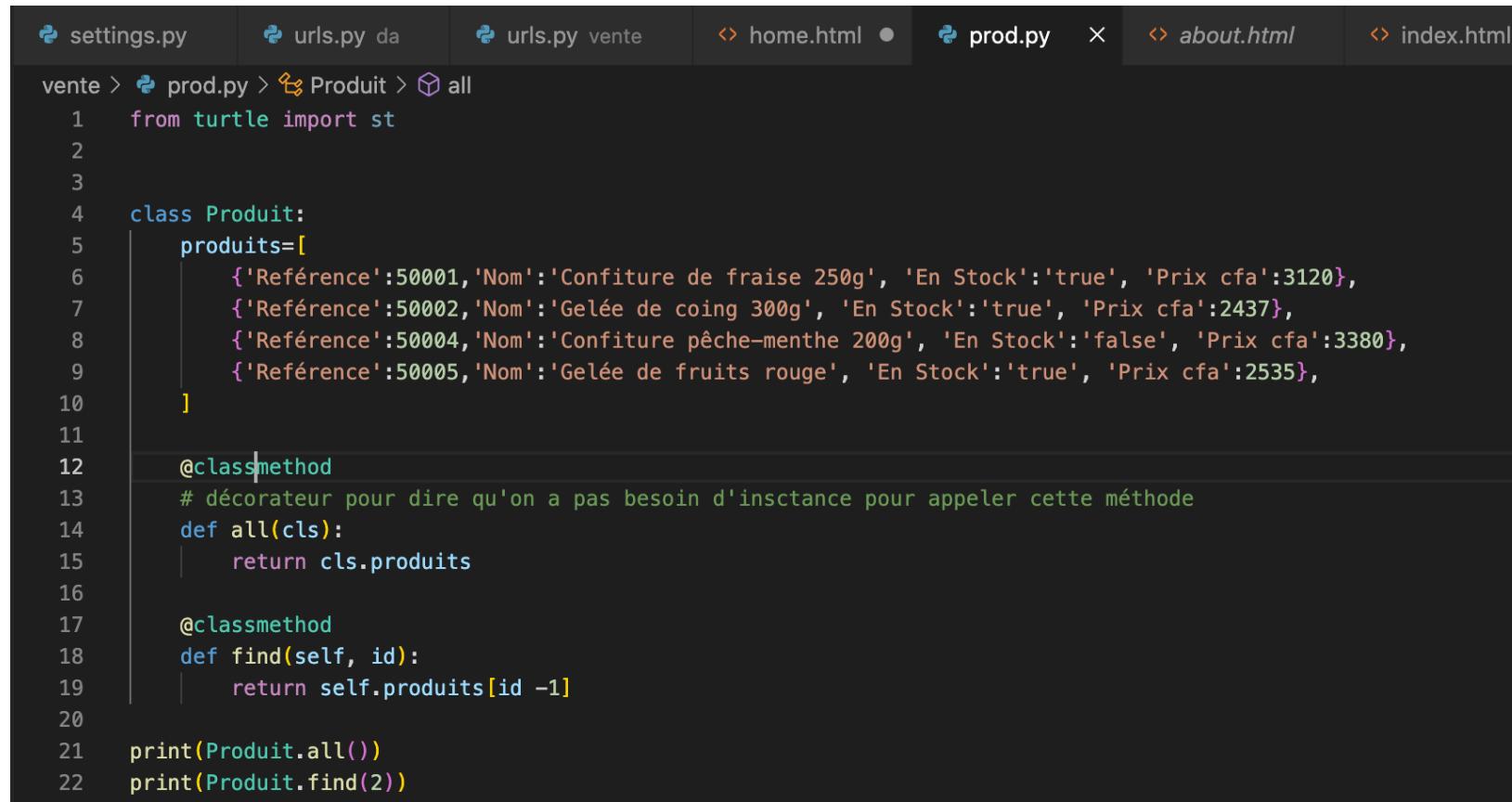
- Remplaçons le # par le lien qui nous permet d'aller à la page des ventes de produits
- Si on clique sur le lien « Visiter nos produits », on doit avoir l'affichage



```
[{"id": 1, "name": "Ventilateur", "description": "Un ventilateur est un appareil destiné à ..."}, {"id": 2, "name": "Ordinateur", "description": "un macbook de marque ..."}, {"id": 3, "name": "Téléphone portable", "description": "un iphone de marque ..."}]
```

Simulation de modèle à travers une classe

■ Créer un fichier prod.py



```
settings.py urls.py da urls.py vente home.html ● prod.py ✘ about.html index.html

vente > prod.py > Produit > all
1  from turtle import st
2
3
4  class Produit:
5      produits=[
6          {'Référence':50001,'Nom':'Confiture de fraise 250g', 'En Stock':'true', 'Prix cfa':3120},
7          {'Référence':50002,'Nom':'Gelée de coing 300g', 'En Stock':'true', 'Prix cfa':2437},
8          {'Référence':50004,'Nom':'Confiture pêche-menthe 200g', 'En Stock':'false', 'Prix cfa':3380},
9          {'Référence':50005,'Nom':'Gelée de fruits rouge', 'En Stock':'true', 'Prix cfa':2535},
10     ]
11
12     @classmethod
13     # décorateur pour dire qu'on a pas besoin d'insctance pour appeler cette méthode
14     def all(cls):
15         return cls.produits
16
17     @classmethod
18     def find(self, id):
19         return self.produits[id -1]
20
21 print(Produit.all())
22 print(Produit.find(2))
```

Simulation de modèle à travers une classe

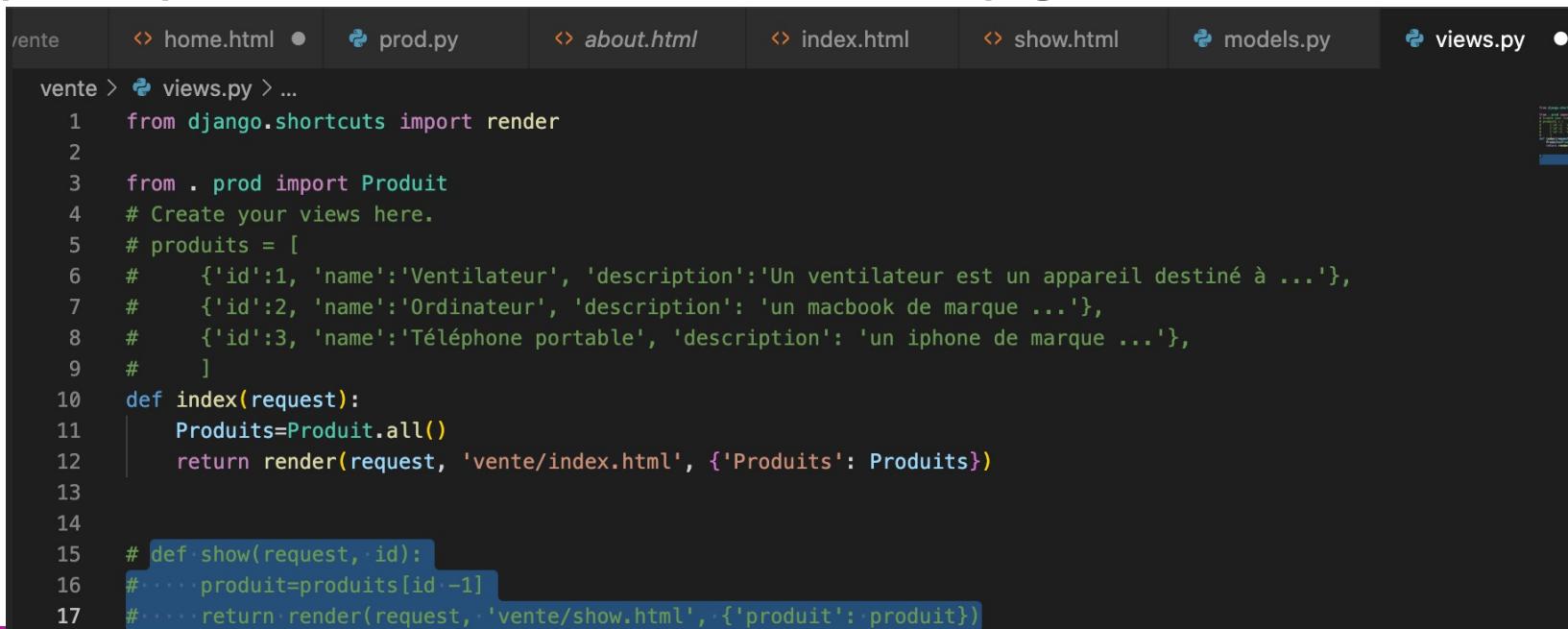
A l'exécution:

```
(gueye) amadoudahirougueye@MBP-de-Amadou da % /Users/amadoudahirougueye/Desktop/Django/gueye/bin/python3.10 /Users/amadoudahirougueye/Desktop/Django/da/vente/prod.py
[{'Référence': 50001, 'Nom': 'Confiture de fraise 250g', 'En Stock': 'true', 'Prix cfa': 3120}, {'Référence': 50002, 'Nom': 'Gelée de coing 300g', 'En Stock': 'true', 'Prix cfa': 2437}, {'Référence': 50004, 'Nom': 'Confiture pêche-menthe 200g', 'En Stock': 'false', 'Prix cfa': 3380}, {'Référence': 50005, 'Nom': 'Gelée de fruits rouge', 'En Stock': 'true', 'Prix cfa': 2535}]
{'Référence': 50002, 'Nom': 'Gelée de coing 300g', 'En Stock': 'true', 'Prix cfa': 2437}
(gueye) amadoudahirougueye@MBP-de-Amadou da %
```

Avec ce code, on a pu lister l'ensemble des produits avec la fonction all() et un seul produit avec la fonction find()

Maintenant, on va aller au niveau de views vente et importer le fichier prod.py

Après on passe la variable Produits au niveau de la page vente/index.html



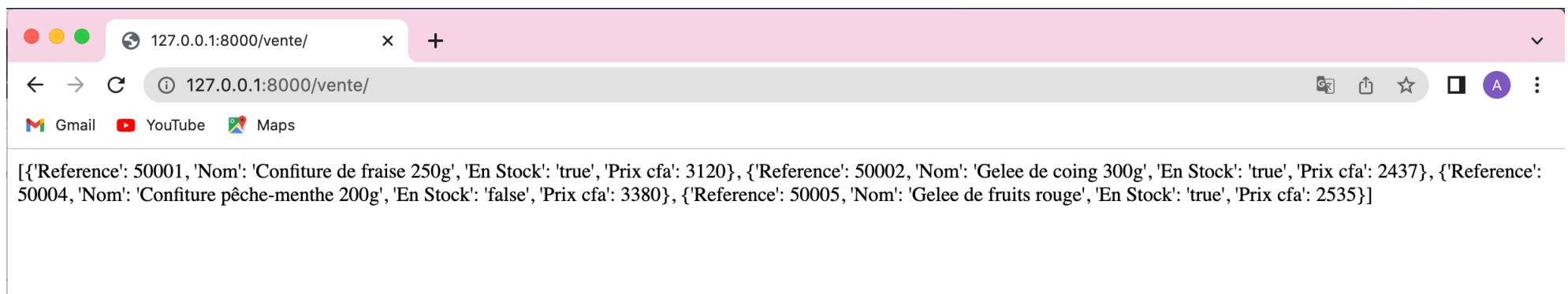
```
vene > views.py > ...
1  from django.shortcuts import render
2
3  from . prod import Produit
4  # Create your views here.
5  # produits = [
6  #     {'id':1, 'name':'Ventilateur', 'description':'Un ventilateur est un appareil destiné à ...'},
7  #     {'id':2, 'name':'Ordinateur', 'description': 'un macbook de marque ...'},
8  #     {'id':3, 'name':'Téléphone portable', 'description': 'un iphone de marque ...'},
9  # ]
10 def index(request):
11     Produits=Produit.all()
12     return render(request, 'vente/index.html', {'Produits': Produits})
13
14
15 # def show(request, id):
16 #     produit=produits[id-1]
17 #     return render(request, 'vente/show.html', {'produit': produit})
```

Simulation de modèle à travers une classe

■ Après on rend disponible la variable Produits dans vente/index.html:

`{{Produits}}`

■ Actualisez la page web et ça donne:



A screenshot of a web browser window. The address bar shows the URL `127.0.0.1:8000/vente/`. The main content area displays a list of dictionaries representing products:

```
[{'Reference': 50001, 'Nom': 'Confiture de fraise 250g', 'En Stock': 'true', 'Prix cfa': 3120}, {'Reference': 50002, 'Nom': 'Gelee de coing 300g', 'En Stock': 'true', 'Prix cfa': 2437}, {'Reference': 50004, 'Nom': 'Confiture pêche-menthe 200g', 'En Stock': 'false', 'Prix cfa': 3380}, {'Reference': 50005, 'Nom': 'Gelee de fruits rouge', 'En Stock': 'true', 'Prix cfa': 2535}]
```

Parcours de la liste des produits et affichage

Ici on veut donner pour chaque produit son nom

Rendons nous dans la page index.html de vente et modifions le ainsi:

```
urls.py vente      home.html ●      prod.py      about.html      index.html X
vente > templates > vente > index.html > ul > li
1   <h1> Accueil produit</h1>
2   <!--{{ Produits }}-->
3   <ul>
4       {%for produit in Produits%}
5           <li> {{produit.Nom}}</li>
6       {%endfor%}
7   </ul>
```

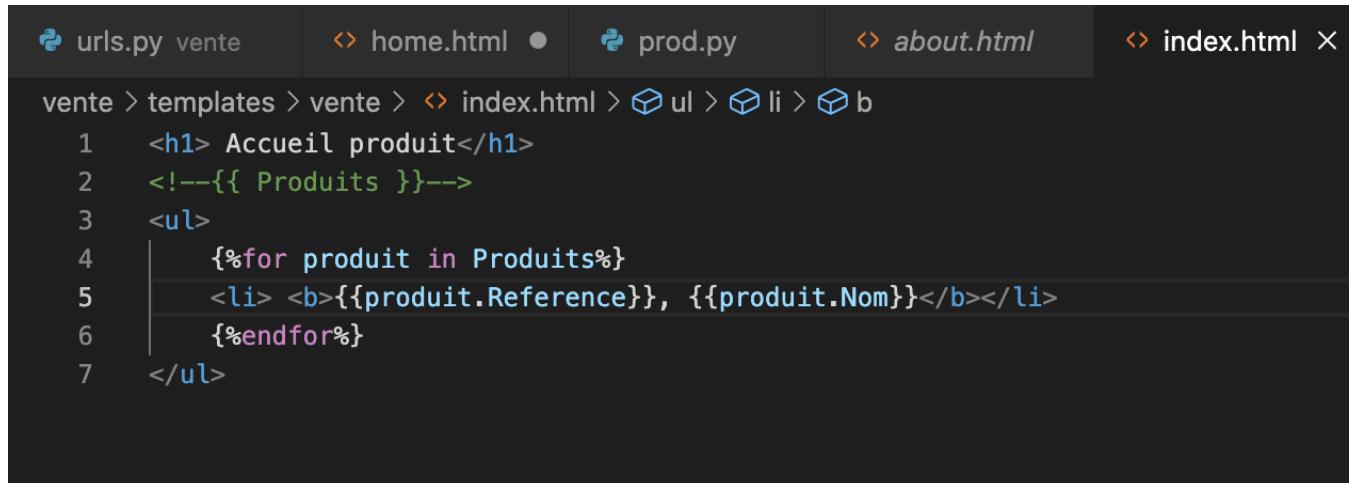
Résultat:

Accueil produit

- Confiture de fraise 250g
- Gelee de coing 300g
- Confiture pêche-menthe 200g
- Gelee de fruits rouge

Parcours de la liste des produits et affichage

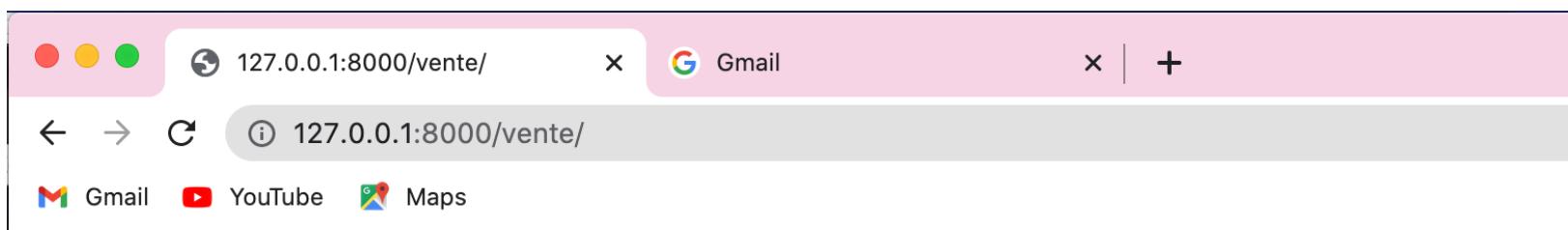
Affichage des références et des noms



The screenshot shows a code editor with several tabs at the top: urls.py vente, home.html, prod.py, about.html, and index.html. The index.html tab is active, showing the following Python template code:

```
1 <h1> Accueil produit</h1>
2 <!--{{ Produits }}-->
3 <ul>
4     {%for produit in Produits%}
5         <li> <b>{{produit.Reference}}, {{produit.Nom}}</b></li>
6     {%endfor%}
7 </ul>
```

Résultat:



Accueil produit

- 50001, Confiture de fraise 250g
- 50002, Gelee de coing 300g
- 50004, Confiture pêche-menthe 200g
- 50005, Gelee de fruits rouge

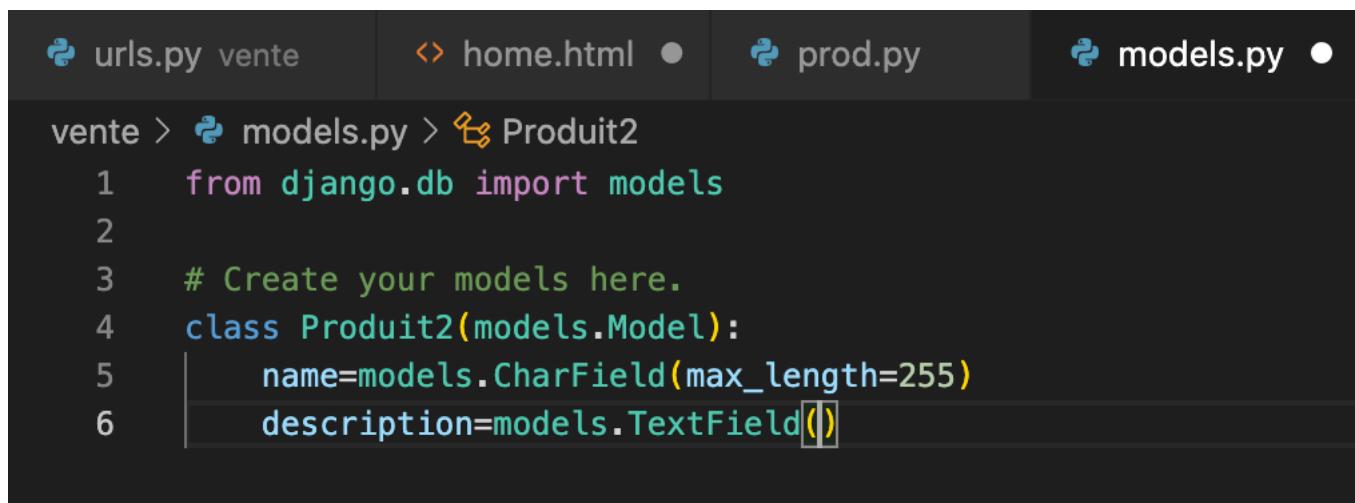
Démarche

Dans cette partie, nous allons voir:

- Comment créer un modèle
- Comment appliquer un modèle et les migrations d'un modèle
- Comment Créer une page d'administration à partir de Django

Création de Modèle

- Pour communiquer avec la base de données SQLite, on aura besoin de créer un modèle à niveau de Django
- C'est au niveau du fichier `models.py` de l'application qu'il faut définir le modèle à l'aide de définitions de classes
- Dans l'exemple qui suit on va créer un modèle qui gère les produits
- Pour cela on va créer dans le modèle une classe pour gérer les produits
- Cette classe comportera deux champs



The screenshot shows a code editor with four tabs at the top: `urls.py`, `home.html`, `prod.py`, and `models.py`. The `models.py` tab is active. Below the tabs, the file structure is shown as `vente > models.py > Produit2`. The code in the editor is:

```
1  from django.db import models
2
3  # Create your models here.
4  class Produit2(models.Model):
5      name=models.CharField(max_length=255)
6      description=models.TextField()
```

Création de Modèle et application migration

- La migration est une relation entre la base de données et le modèle
- Après avoir créé le modèle, on a besoin d'appliquer les migrations dans la base sqlite
- On quitte le serveur par CTL+C puis on applique la migration
- Pour voir les migrations possibles, tapez la commande **./manage.py showmigrations**
- Tapons **./manage.py makemigrations**

```
-----  
:(gueye) amadoudahirougueye@MBP-de-Amadou da % ./manage.py makemigrations      ]  
[{'Reference': 50001, 'Nom': 'Confiture de fraise 250g', 'En Stock': 'true', 'Pr  
ix cfa': 3120}, {'Reference': 50002, 'Nom': 'Gelee de coing 300g', 'En Stock': 'true', 'Pri  
x cfa': 2437}, {'Reference': 50004, 'Nom': 'Confiture pêche-menthe 20  
0g', 'En Stock': 'false', 'Prix cfa': 3380}, {'Reference': 50005, 'Nom': 'Gelee  
de fruits rouge', 'En Stock': 'true', 'Prix cfa': 2535}]  
{'Reference': 50002, 'Nom': 'Gelee de coing 300g', 'En Stock': 'true', 'Prix cfa  
': 2437}  
Migrations for 'vente':  
  vente/migrations/0001_initial.py  
    - Create model Produit2  
(gueye) amadoudahirougueye@MBP-de-Amadou da %
```

- On remarque que:

- Il y a une migration pour la vente
- un fichier 0001_initial.py est créé
- Il a créé le modèle Produits2

Création de Modèle et application migration

Voici un extrait du fichier 0001_initial.py

```
13     operations = [
14         migrations.CreateModel(
15             name='Produit2',
16             fields=[
17                 ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
18                 ('name', models.CharField(max_length=255)),
19                 ('description', models.TextField()),
20             ],
21         ),
22     ]
```

Maintenant on applique la migration par la commande ./manage.py migrate

```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, vente
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
  Applying vente.0001_initial... OK
(gueye) amadoudahirogueye@MBP-de-Amadou da %
```

Création de Modèle et application migration

■ Pour la base de données, il faut installer SQLite Viewer pour voir le contenu

■ Un clic sur db.sqlite3 nous permet de voir en bas la table `vente_Produit2` créée

The screenshot shows the SQLite Viewer interface with the database file 'db.sqlite3' open. The left sidebar displays the project structure with files like __pycache__, __init__.py, asgi.py, settings.py, urls.py, views.py, wsgi.py, templates, vente, migrations, templates/vente, and various Python files. The main pane shows a table with 19 records. The columns are id, app, name, and applied. The 'app' column contains values like contenttypes, auth, admin, and auth again. The 'name' column contains names like '0001_initial' through '0007_alter_validator...'. The 'applied' column shows dates from 2022-09-14. At the bottom of the table, there is a row for the 'vente_produit2' table, which has 12 records. The 'app' column for these rows is 'auth'. The bottom of the viewer shows a terminal window with a Python traceback and a Jupyter variables panel.

id	app	name	applied
1	contenttypes	0001_initial	2022-09-14 12:16:0...
2	auth	0001_initial	2022-09-14 12:16:0...
3	admin	0001_initial	2022-09-14 12:16:0...
4	admin	0002_logentry_remo...	2022-09-14 12:16:0...
5	admin	0003_logentry_add_...	2022-09-14 12:16:0...
6	contenttypes	0002_remove_content...	2022-09-14 12:16:0...
7	auth	0002_alter_permission...	2022-09-14 12:16:0...
8	auth	0003_alter_user_email...	2022-09-14 12:16:0...
9	auth	0004_alter_user_us...	2022-09-14 12:16:0...
10	auth	0005_alter_user_last...	2022-09-14 12:16:0...
11	auth	0006_require_contentte...	2022-09-14 12:16:0...
12	auth	0007_alter_validator...	2022-09-14 12:16:0...

```
80}, {'Référence': 50005, 'Nom': 'Gelée de fruits rouge', 'En Stock': 'true', 'Prix cfa': 2535}]  
{'Référence': 50002, 'Nom': 'Gelée de coing 300g', 'En Stock': 'true', 'Prix cfa': 2437}  
(gueye) amadoudahirougueye@MBP-de-Amadou da % /Users/amadoudahirougueye/Desktop/Django/gueye/bin/python3.10 /Users/amadoudahirougueye/Desktop/Django/da/vente/prod.py  
Traceback (most recent call last):  
  File "/Users/amadoudahirougueye/Desktop/Django/da/vente/prod.py", line 21, in <module>  
    print(Produit.all())  
TypeError: Produit.all() missing 1 required positional argument: 'cls'
```

Création de Modèle et application migration

■ Un clic sur **vente_Produit2** donne

The screenshot shows a SQLite database browser interface with the following details:

- Database: db.sqlite3
- Tables: 12
- Selected Table: vente_produit2
- Table Columns (Visible): id, name, description
- Table Columns (Searchable): Search column... (for id, name, description)

■ Nous remarquons que l'iD est créé automatiquement par Django

Remplissage de la table: avec le terminal

- La commande shell de ./manage.py nous permet d'avoir un terminal interactif pour le python

```
[gueye] amadoudahirogueye@MBP-de-Amadou da % ./manage.py shell
Python 3.10.5 (v3.10.5:f377153967, Jun  6 2022, 12:36:10) [Clang 13.0.0 (clang-1
300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> 
```

- Pour créer un produit, on importe d'abord la classe Produit2

```
[>>> from vente.models import Produit2
[>>> p=Produit2()
[>>> p.name='Ordinateur'
[>>> p.description='Ordinateur macbook de marque'
[>>> p.save()
>>> 
```

- Actualisons la base de données pour voir l'enregistrement

db.sqlite3					
		Search tables... Reset Filters Records: 1			
Tables (12)		id	name	description	
		Search column...	Search column...	Search column...	
>	django_migrations	1	1	Ordinateur	Ordinateur macbook de marque
>	sqlite_sequence				
>	auth_group_permissions				
>	auth_user_groups				
>	auth_user_user_perm				
>	django_admin_log				
>	django_content_type				
>	auth_permission				
>	auth_group				

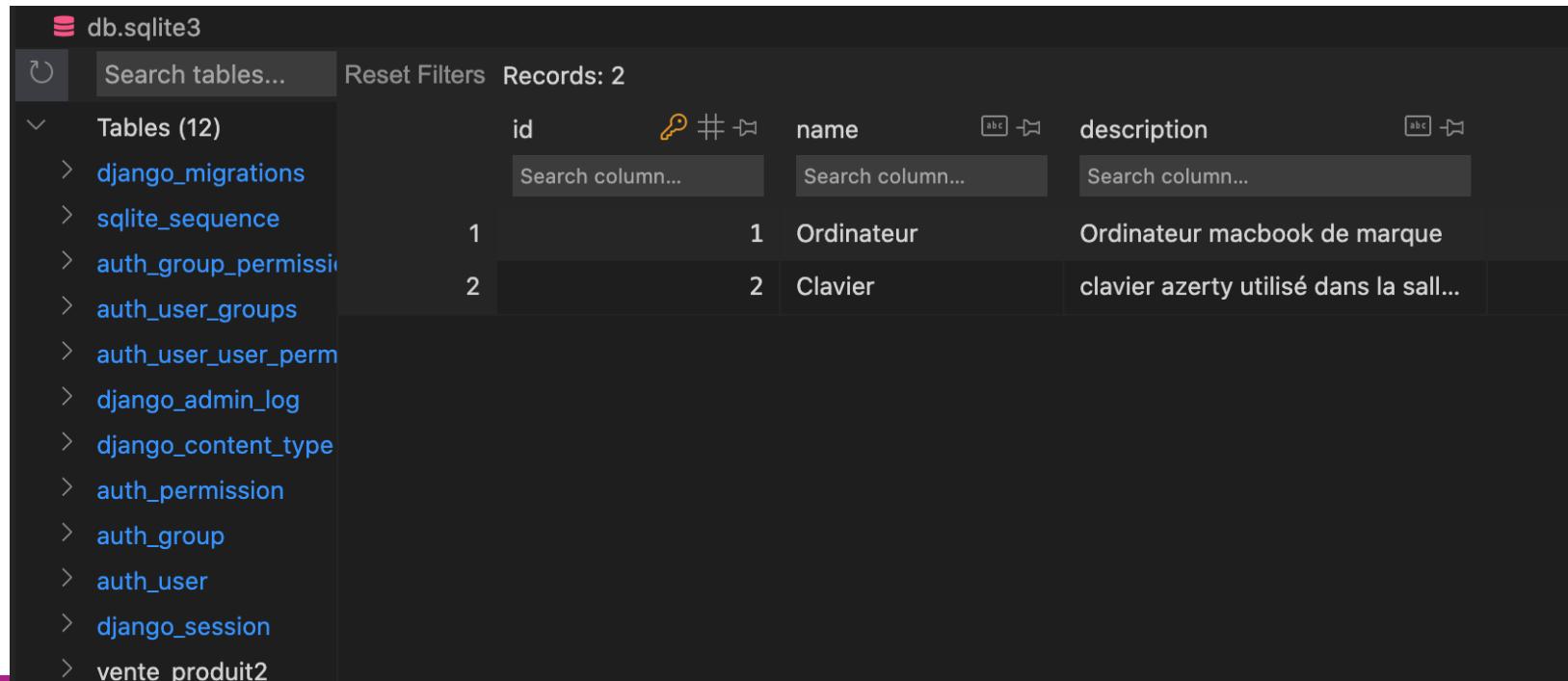
Remplissage de la table: avec le terminal

■ Ici, au lieu d'utiliser des requêtes pour la base de données, Django fait du ORM (Object Relational Mapping) autrement dit on fait du mappage au niveau de la base de données

■ On peut aussi créer l'objet avec des arguments:

```
[>>> p=Produit2(name='Clavier', description='clavier azerty utilisé dans la salle master SI-SR UADB')
[>>> p.save()
>>> ]
```

■ Si on actualise la BD:



db.sqlite3					
		Search tables...	Reset Filters	Records: 2	
Tables (12)					
>	django_migrations				
>	sqlite_sequence				
>	auth_group_permissions				
>	auth_user_groups				
>	auth_user_user_perm				
>	django_admin_log				
>	django_content_type				
>	auth_permission				
>	auth_group				
>	auth_user				
>	django_session				
>	vente_produit2				

id	name	description
1	Ordinateur	Ordinateur macbook de marque
2	Clavier	clavier azerty utilisé dans la sall...

Remplissage de la table: avec le terminal

■ Autre méthode de créer un produit dans la base de données à l'aide de la fonction `create`

```
[>>> Produit2.objects.create(name='Raspberry pi', description='mini ordoninateur')
)
<Produit2: Produit2 object (3)>
>>>
```

■ Il va créer un objet `Produit2` et le sauvegarder directement dans la base (pas besoin de `save`)

■ Rq: le troisième produit est tapé deux fois, ce qui fait qu'on a 4 produits

■ Si on actualise:

The screenshot shows the SQLite browser interface with the database file "db.sqlite3" selected. On the left, a tree view lists tables: "Tables (12)" and "Produit2". The "Produit2" table is expanded, showing its columns: id, name, and description. The table contains 4 records:

		id	name	description
		1	Ordinateur	Ordinateur macbook de marque
		2	Clavier	clavier azerty utilisé dans la salle
		3	Raspberry pi	mini ordoninateur
		4	Raspberry pi	mini ordoninateur

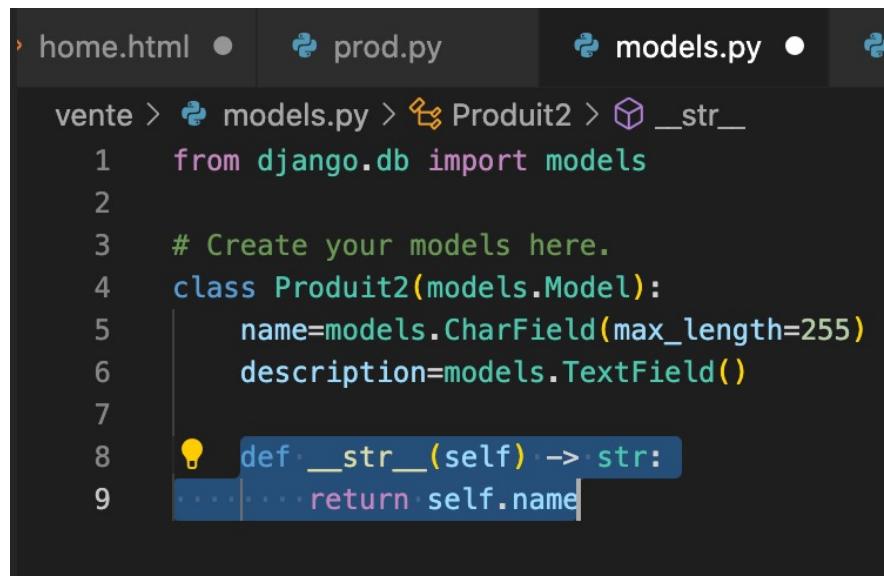
Récupération résultat

- Pour récupérer tous les produits:

```
[>>> Produit2.objects.all()
[<Produit2: Produit2 object (1)>, <Produit2: Produit2 object (2)>, <Produit2: Produit2 object (3)>, <Produit2: Produit2 object (4)>]
>>>
```

- On voit cette forme d'affichage des produits n'est pas idéale

- On va changer l'affichage au niveau du modèle grâce à la méthode spéciale `__str__`



```
home.html ● prod.py ● models.py ●
 vente > models.py > Produit2 > __str__
1 from django.db import models
2
3 # Create your models here.
4 class Produit2(models.Model):
5     name=models.CharField(max_length=255)
6     description=models.TextField()
7
8     def __str__(self) -> str:
9         return self.name
```

Récupération résultat

- Pour récupérer tous les produits, on utilise **all** et pour un produit on utilise **get**

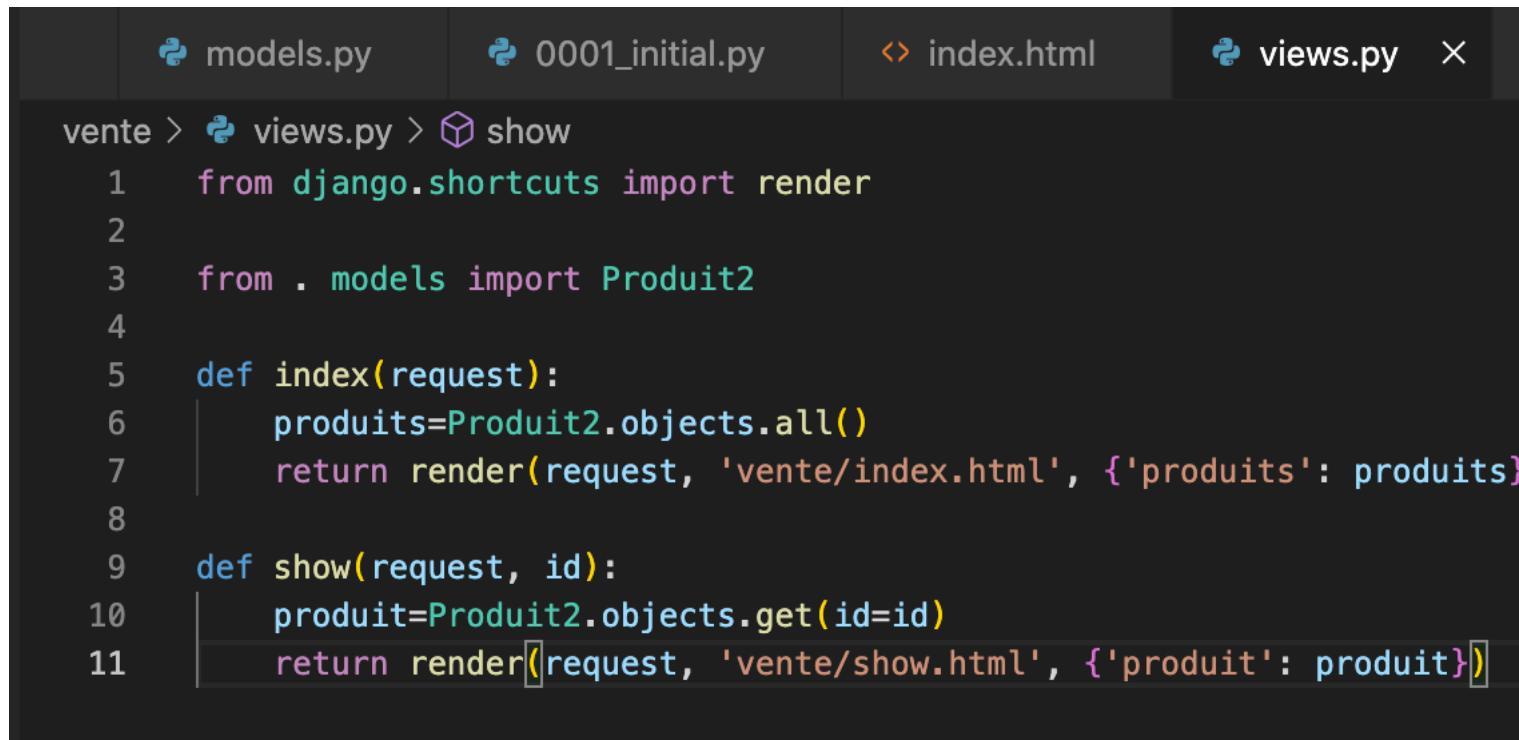
```
[>>> from vente.models import Produit2
[>>> Produit2.objects.all()
[>>> <QuerySet [<Produit2: Ordinateur>, <Produit2: Clavier>, <Produit2: Raspberry pi>
, <Produit2: Raspberry pi>]>
[>>> Produit2.objects.get(id=1)
[>>> <Produit2: Ordinateur>
[>>> ]]
```

- Récupérer le primary key pk

```
[>>> p=Produit2.objects.get(pk=1)
[>>> p
<Produit2: Ordinateur>
[>>> p.name
'Ordinateur'
[>>> p.description
'Ordinateur macbook de marque'
[>>> ]]
```

Affichage via une page web

■ Pour cela, rendons nous dans le fichier views.py de l'application vente et modifions le a



The screenshot shows a code editor with a dark theme. The top navigation bar includes tabs for 'models.py', '0001_initial.py', 'index.html', and 'views.py'. Below the tabs, the code editor displays the 'views.py' file for the 'vente' application. The code contains two functions: 'index' and 'show'. The 'index' function retrieves all products from the 'Produit2' model and renders them using 'index.html'. The 'show' function retrieves a specific product by its ID and renders it using 'show.html'.

```
models.py 0001_initial.py index.html views.py ×

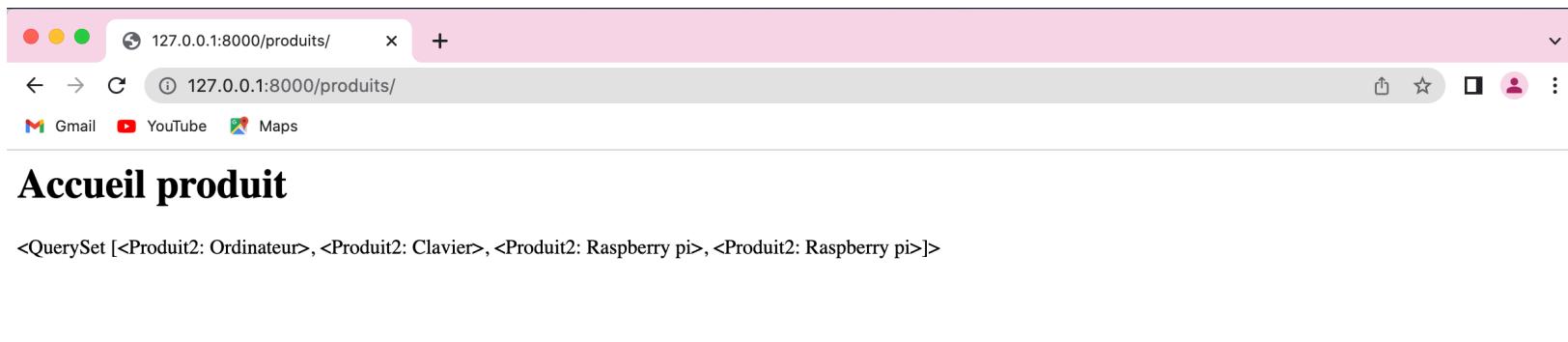
vente > views.py > show
1  from django.shortcuts import render
2
3  from .models import Produit2
4
5  def index(request):
6      produits=Produit2.objects.all()
7      return render(request, 'vente/index.html', {'produits': produits})
8
9  def show(request, id):
10     produit=Produit2.objects.get(id=id)
11     return render(request, 'vente/show.html', {'produit': produit})
```

■ Redemarrons le serveur web : ./manger.py runserver

■ Ne pas oublier de passer les variables contextes dans les fichiers index.html et show.ht

Affichage via une page web

Résultat 1



Depuis la ligne de commande, on peut ajouter un nouveau produit

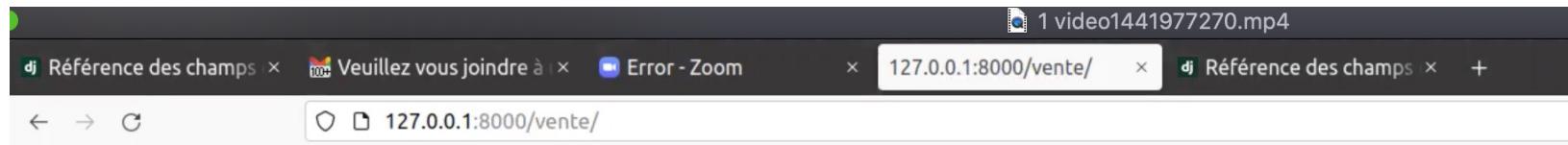
```
[>>> from vente.models import Produit2
[>>> Produit2.objects.create(name='Figo', description='Figo bar moins cher')
<Produit2: Figo>
>>> ]
```

Puis on redémarre le serveur web:



TP

■ Reprendre tout ce qu'on a fait jusque là et l'avoir maintenant sous cette forme d'affichage



1 Ordinateur

Notre ordinateur de ...

[Lire la suite](#)

2 Clavier

AZERTY

[Lire la suite](#)

3 Hardware

Materiel

[Lire la suite](#)

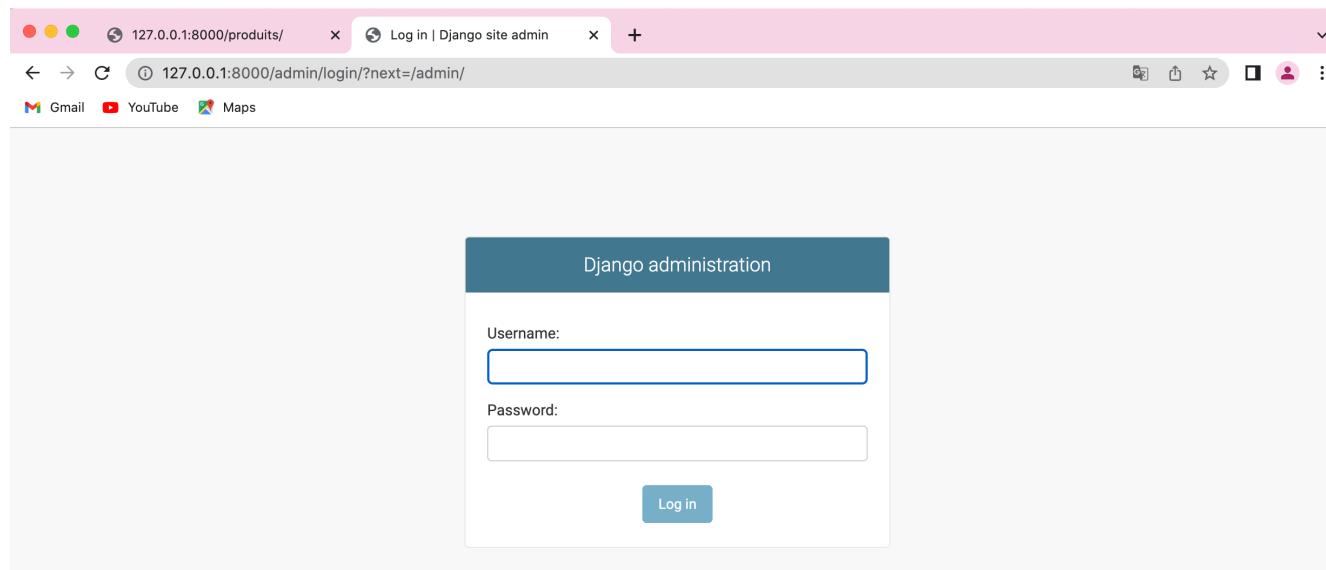
4 Frigo

Frigo barre

[Lire la suite](#)

Page d'administration de Django

- Le côté fort de Django c'est sa page d'administration
- Avec Django, pas besoin de créer une page d'administration contrairement à d'autres frameworks
- Si on tape le lien `127.0.0.1:8000/admin` on a:

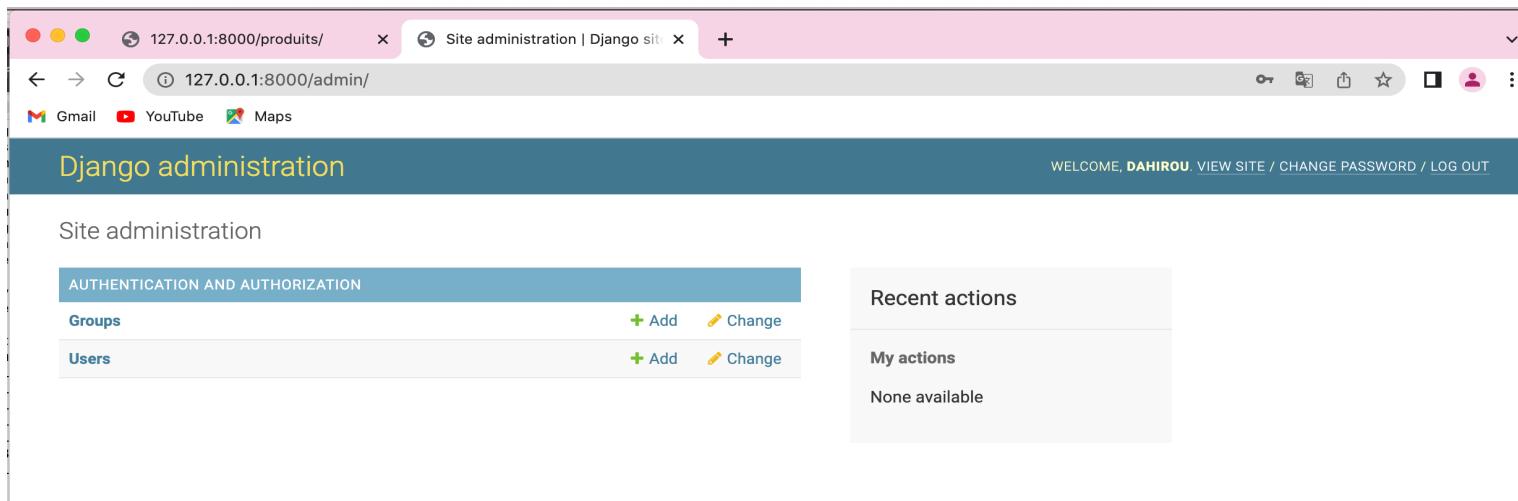


- Au niveau de `manage.py`, on a une commande `createsuperuser`
- On va taper cette commande pour créer un utilisateur

Page d'administration de Django

```
(gueye) amadoudahirougueye@MBP-de-Amadou da % ./manage.py createsuperuser
Username (leave blank to use 'amadoudahirougueye'): dahirou
Email address: g.dahirou@gmail.com
[Password]:
[Password (again)]:
Superuser created successfully.
(gueye) amadoudahirougueye@MBP-de-Amadou da %
```

- On redémarre le serveur et à partir de là, on peut donner le login et le mot de passe pour avoir accès à la page d'administration de Django



- A partir de là, on peut créer d'autres utilisateurs
- On veut vérifier les produits à partir de la page d'administration

Mise à disposition de la page d'administration des modèles créés pour chaque application

- La création d'une application sous Django entraîne la création automatique d'un dossier admin.py
- On importe d'abord la classe Produit2
- Pour amener le modèle Produit2 dans la page d'administration, on fait ceci:

```
prod.py      models.py      admin.py ×  
vente > admin.py  
1   from django.contrib import admin  
2   from .models import Produit2  
3   # Register your models here.  
4  
5   admin.site.register(Produit2)  
6
```

- Si on actualise la page web

The screenshot shows the Django administration site at 127.0.0.1:8000/admin/. The top navigation bar includes links for Gmail, YouTube, and Maps. The main header says "Django administration" and "WELCOME, DAHIROU. VIEW SITE / CHANGE PASSWORD / LOG OUT". The left sidebar lists "Site administration" and two main sections: "AUTHENTICATION AND AUTHORIZATION" (Groups, Users) and "VENTE" (Produit2). The "VENTE" section has "Produit2" listed with "Add" and "Change" buttons. A "Recent actions" sidebar on the right shows "My actions: None available".

Mise à disposition de la page d'administration des modèles créés pour chaque application

■ Un clic sur Produit2s nous donne:

The screenshot shows the Django admin interface at the URL 127.0.0.1:8000/admin/vente/produit2/. The left sidebar has a 'VENTE' section with 'Produit2s' selected. The main content area displays a list of products with checkboxes next to them. The products listed are: PRODUIT2, Frigo, Raspberry pi, Raspberry pi, Clavier, and Ordinateur. A total of 5 produit2s are shown.

■ A partir de là, on peut apporter des modifications depuis la page d'administration et qui se répercuteront directement dans la base et vice-versa

TP à réaliser

- Créer une application Etudiant avec un modèle Etudiant qui contient les caractéristiques: code_permanent, prenom, nom, adresse, image et date de naissance.
- NB pour l'image, il faut installer pillow par pip install pillow
- Après cela, créer une migration avec la commande ./manage.py makemigrations
- Après cela, on doit avoir une migration qui est appliquée, on doit donc la sauvegarder par la commande ./manage.py migrate
- Vérifier au niveau de la base de données sqlite si vous avez la table etudiant_etudiant
- Pour ajouter des étudiants, vous allez le faire à partir de la page d'administration
 - Pour cela, aller dans le fichier admin.py de l'application etudiant et refaire le scénario précédent en important la classe etudiant du modèle *from .models import Etudiant* et on passe par l'administration en faisant: *admin.site.register(Etudiant)*
 - Redémarrer le serveur web
 - Et on doit apercevoir ça sur le site

TP à réaliser

■ L'interface d'ajout d'un étudiant dans la page d'administration doit se présenter ainsi:

Django administration

Home > Etudiant > Etudiants - Add etudiant

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

Groups + Add
Users + Add

ETUDIANT

Etudiants + Add

VENTE

Produits + Add

Add etudiant

Code permanent: 436543

Prenom: Malal

Nom: Dem

Adresse: Bambey

Date naiss: 1997-01-02 Today |

Image: No file selected.

■ Pour avoir un affichage avec prénom nom, on peut ajouter cette fonction dans le modèle:

Def __str__(sel) ->:

```
return self.prenom+ ' ' self.nom
```

Ou

Def __str__(sel) ->:

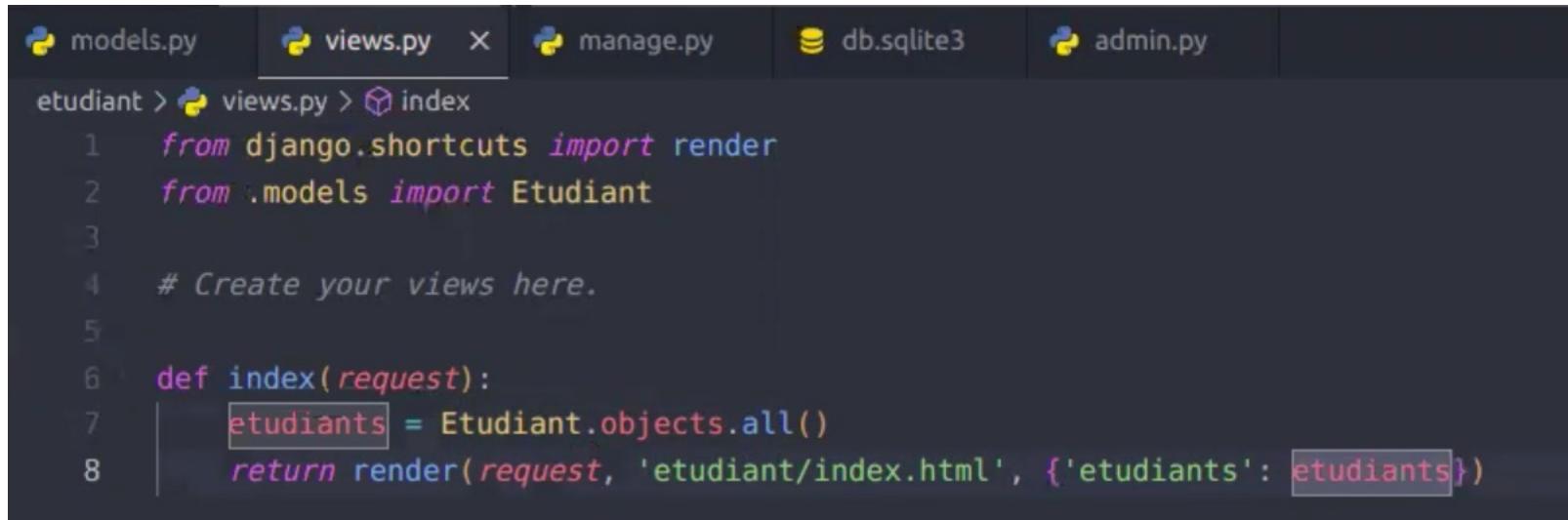
```
return "{}{}" .format(self.prenom, self.nom)
```

TP à réaliser

■ Puis afficher les étudiants saisis dans la page d'administraton

■ Maintenant pour afficher les étudiants dans la page web

- Aller au niveau de views de l'application étudiant et passer les étudiants au niveau de la page etudiant/index.html
- Commencer d'abord Importer la classe Etudiant de models
- Recupérer tous les étudiants à travers objects.all et on le passe au niveau de la page etudiant/index.html
- Voila ce à quoi ça ressemble

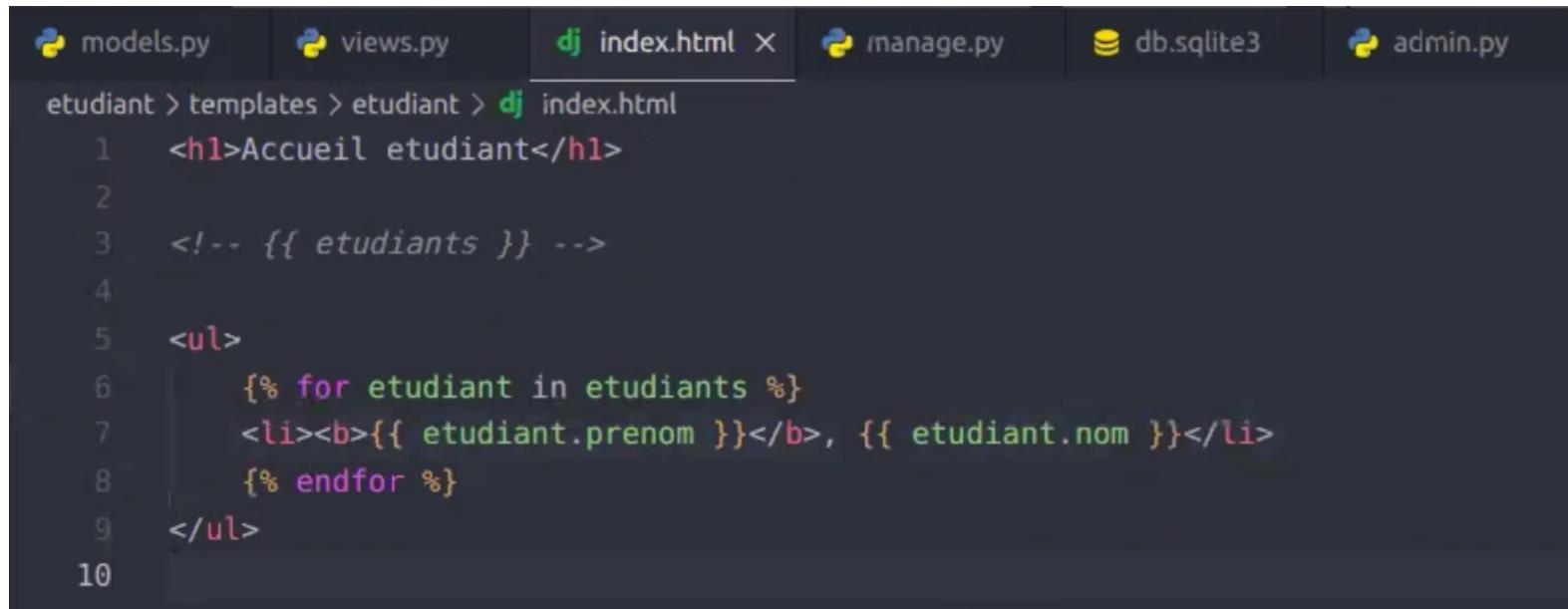


The screenshot shows a code editor with a dark theme. The tab bar at the top includes 'models.py', 'views.py', 'manage.py', 'db.sqlite3', and 'admin.py'. The 'views.py' tab is active. Below the tabs, the file path 'etudiant > views.py > index' is displayed. The code itself is as follows:

```
1  from django.shortcuts import render
2  from .models import Etudiant
3
4  # Create your views here.
5
6  def index(request):
7      etudiants = Etudiant.objects.all()
8      return render(request, 'etudiant/index.html', {'etudiants': etudiants})
```

TP à réaliser

■ Au niveau du fichier index.html, parcourez les étudiants pour les afficher



```
models.py      views.py      dj index.html X      manage.py      db.sqlite3      admin.py
etudiant > templates > etudiant > dj index.html
1  <h1>Accueil etudiant</h1>
2
3  <!-- {{ etudiants }} -->
4
5  <ul>
6      {% for etudiant in etudiants %}
7          <li><b>{{ etudiant.prenom }}</b>, {{ etudiant.nom }}</li>
8      {% endfor %}
9  </ul>
10
```

■ Actualisez la page avec l'URL: adresse:8000/etudiant

Résultat d'affichage??????

FIN

Projet sous Django

Pr ADG – UADB 2022