Introduction au Web et Interface Homme/Machine

Analyse et synthèse HTML en Python

Semestre 2 – Guillaume CLEUZIOU – Romuald THION

HTTP et API DOM en Python

HTTP en Python

https://requests.readthedocs.io/en/latest/

- Principal module Python pour HTTP
- Permet programmatiquement :
  - De créer des **requêtes**
  - Manipuler les **en-têtes**
  - Recevoir les **réponses** synchrones (bloquantes)
    - Pour asynchrones (non-bloquant), voir aiohttp !
  - Accéder au contenu (*body*) et le **parser**

Exemple request

- `import requests` pour charger le module
- Permet d'accéder/manipuler toutes les informations qu'on voit dans les DevTools
- Ne sait **pas interpréter/parser le HTML**
  - Reste du texte brut

```
>>> r = requests.get('https://api.github.com/user', auth=('user', 'pass'))
>>> r.status_code
200
>>> r.headers['content-type']
'application/json; charset=utf8'
>>> r.encoding
'utf-8'
>>> r.text
'{"type":"User"...'
>>> r.json()
{'private_gists': 419, 'total_private_repos': 77, ...}
```

La représentation HTML en mémoire

le Document Object Model (DOM) est une Application Programming Interface (API) standard

- cette interface permet d'interagir programmatiquement avec les contenus en mémoire dans le navigateur
- exemples : *documents HTML*, XML, SVG, MathML etc.
- tous les navigateurs proposent une implémentation JavaScript conforme à cette API, mais elle est aussi disponible pour d'autres langages, dont Python

Exemple en JS dans le navigateur

Avec le site https://insight.nc/

```
// le <div> du pied de page
const footer = document.getElementById('footer-info')

// accès en lecture ET écriture (live) aux attributs
footer.innerText = 'Texte modifié'

// les <img> de la page avec la classe size-large
const images = document.querySelectorAll('img.size-large')
// affichage de toutes les sources
for (const img of images)
    console.log(img.src)
```

HTML en Python

Module requests-html

- `Chargement du module`
  - `from requests_html import HTMLSession`
- Ce module combine :
  - requests pour HTTP
  - Beautiful Soup 4 pour le *parsing* HTML
    - C'est-à-dire l'analyse du HTML et la production d'un document type DOM en mémoire
- Attention, ce n'est **pas le DOM standard**, mais ça ressemble
  - Plus facile d'utilisation, plus « python »

Exemple (1/2)

```
from requests_html import HTMLSession
session = HTMLSession()
resp = session.get("https://insight.nc/?page_id=1194")

# le document brut, comme avec requests
resp.text

# le document HTML analysé
resp.html
```

Exemple (2/2)

```
# comme avec le DOM en JS
resp.html.find('#footer-info')
#[<Element 'div' id='footer-info'>]

images = resp.html.find("img.size-large")
for img in images:
    print(img.attrs["src"])
#https://insight.nc/[...]Fidji_Insight-1024x576.jpg
#https://insight.nc/[...]Voh-NC_Insight-1024x576.jpg
#...[]
```

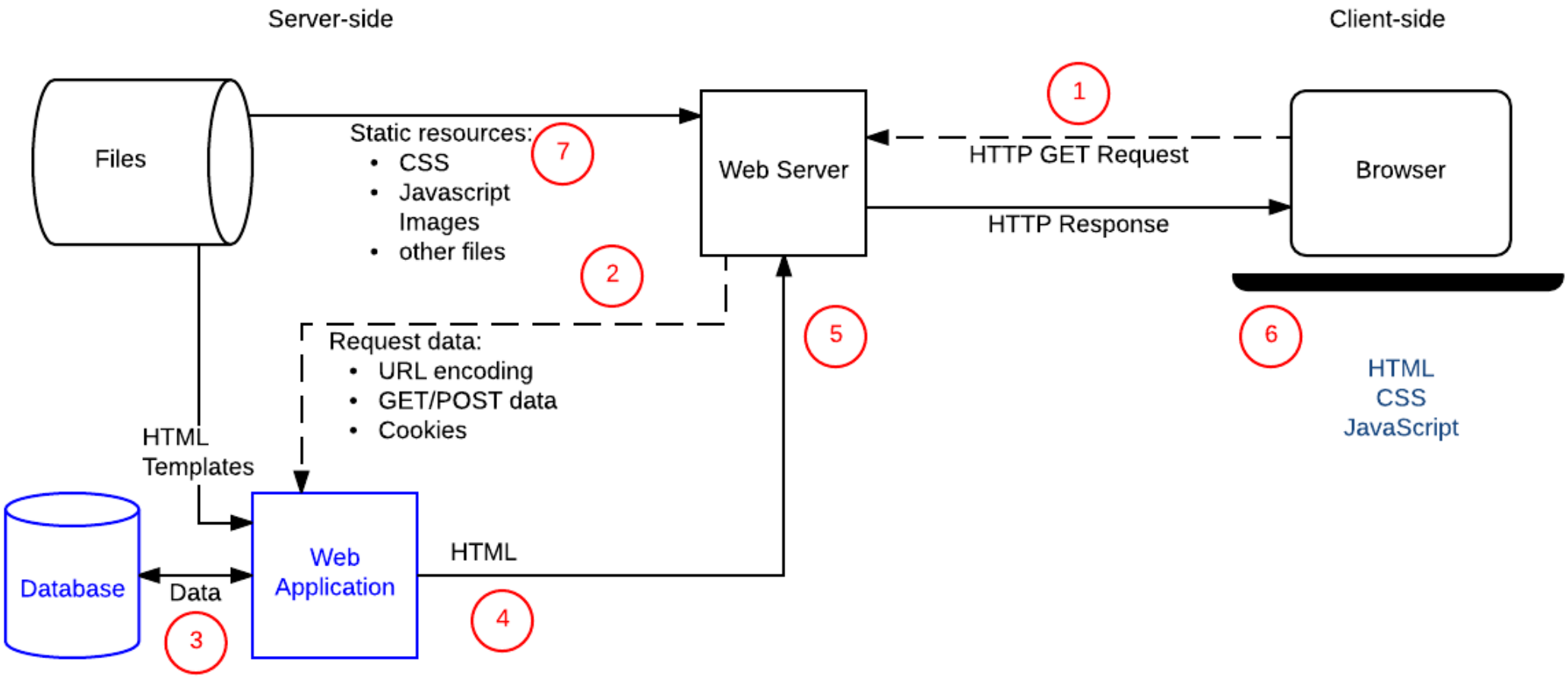
La génération dynamique de pages HTML

Génération automatique de pages HTML

Motivations

- La plupart des pages web présentes sur la toile n'ont pas été écrites entièrement « à la main »
  - Les contenus sont stockés dans une **base de données**
  - Les pages HTML sont **générées** à partir généralement d'un modèle fixe auquel on intègre des contenus extraits de la base
- Le contenu HTML peut être généré automatiquement par un programme
  - Côté client (le navigateur, dit aussi *front*) :
    - en JavaScript, à partir de contenus « bruts » envoyés par le serveur
  - Côté serveur (dit *back*), par un serveur web dit **serveur d'application** :
    - en PhP, en Python, en Java, en JavaScript (Node.js)...

Architecture classique n-tiers d'application web



https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Client-Server_overview#anatomy_of_a_dynamic_request

Génération automatique de pages HTML

Génération automatique de pages HTML

Principe général

Sur le même principe de séparation :

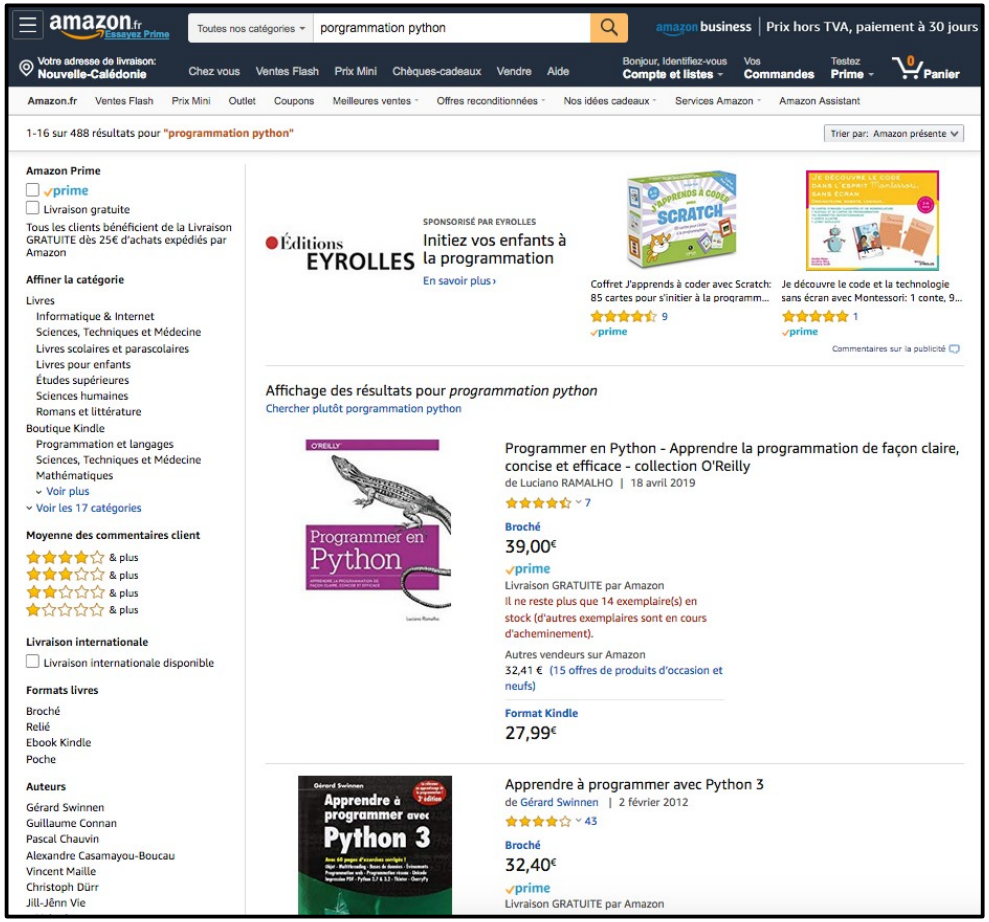
structure+contenu (HTML) vs. mise en forme (CSS)

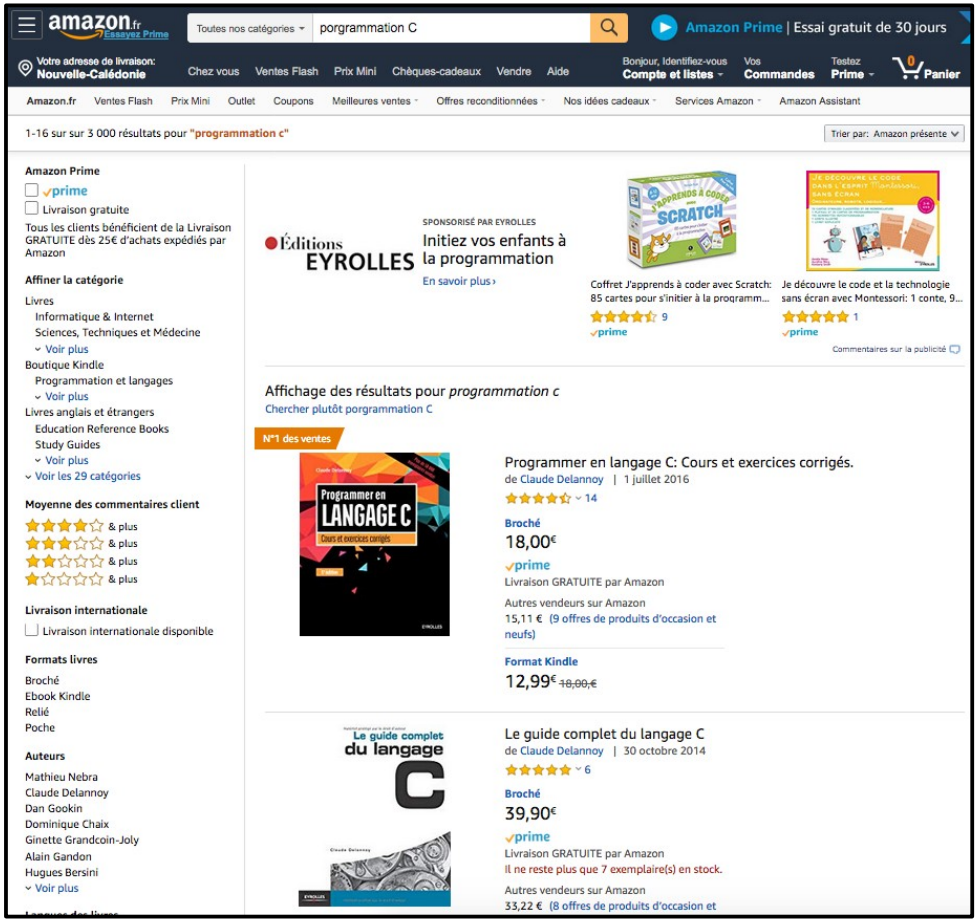
Il sera utile de découper (encore) notre fichier HTML en séparant :

- **Ce qui est commun** : à plusieurs pages ou aux versions d'une même page (généralement la structure globale, quelques titres, en-tête, etc.)
  - On peut enregistrer ce modèle dans un fichier une bonne fois pour toutes : le contenu **est statique**
- Ce qui est **susceptible de changer** : d'une page (ou version) à une autre (généralement le contenu affiché : textes, images, liens, etc.)
  -

Génération automatique de pages HTML

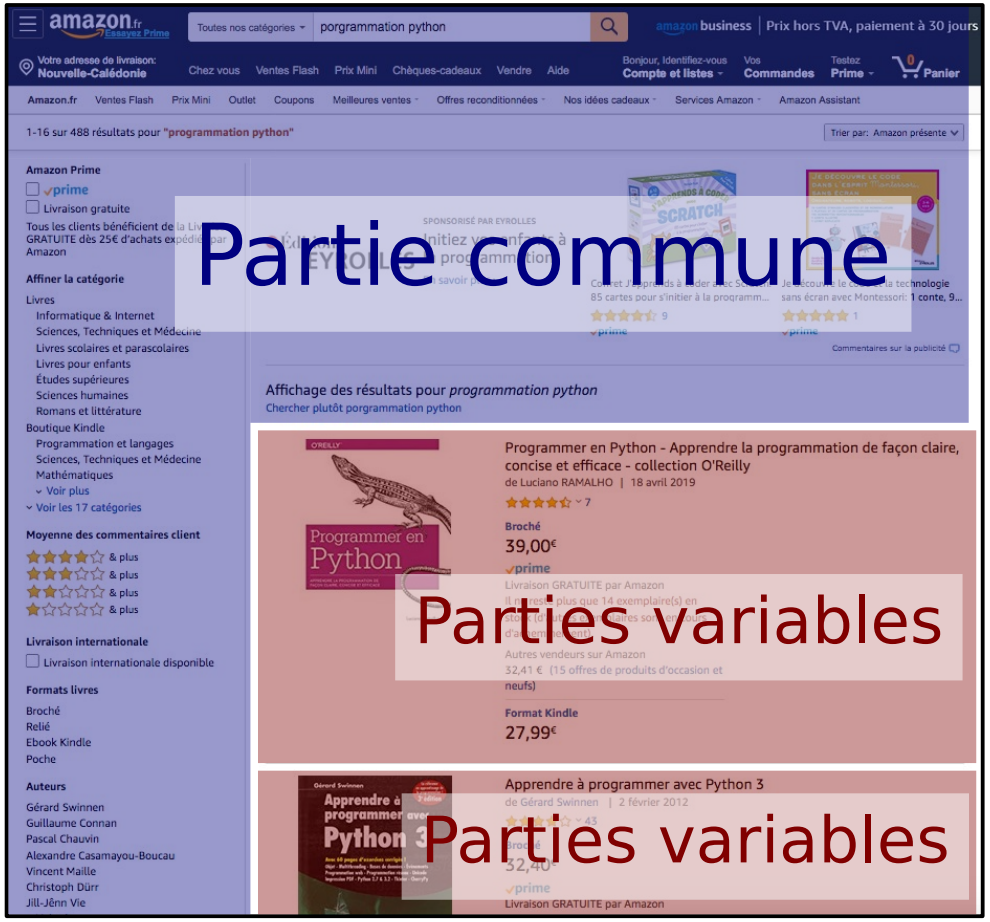
Découpage structure vs. contenu

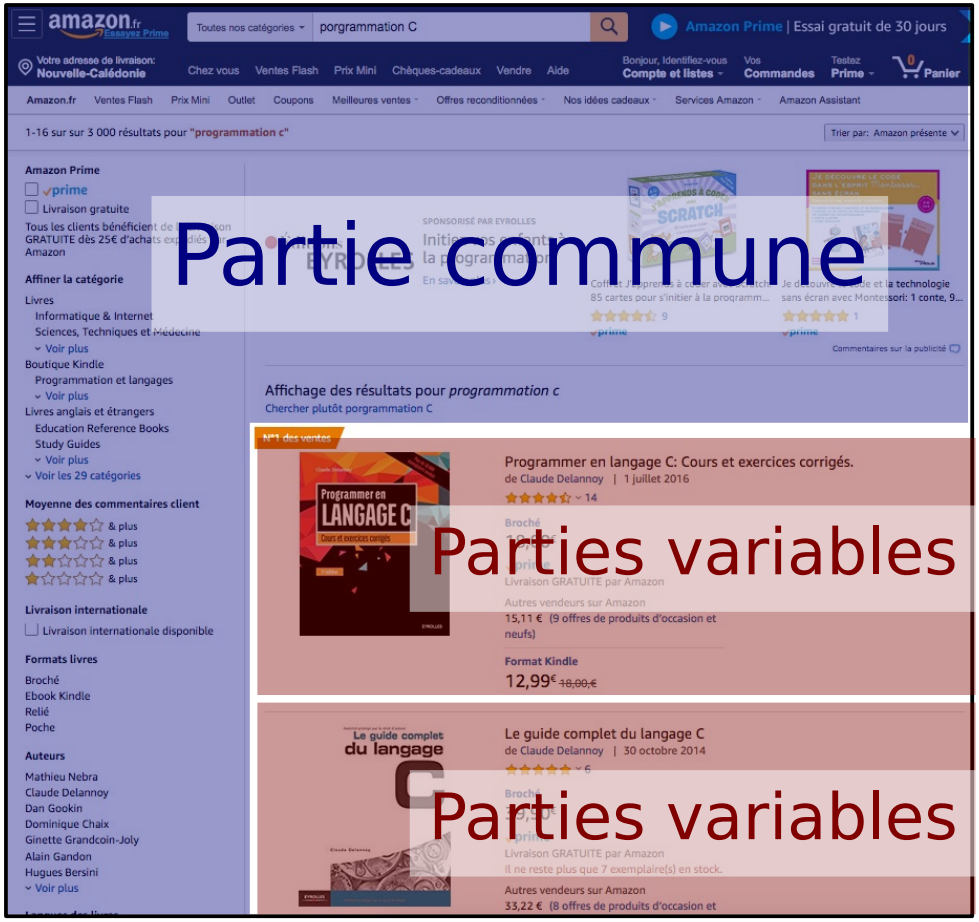




Génération automatique de pages HTML

Découpage structure vs. contenu





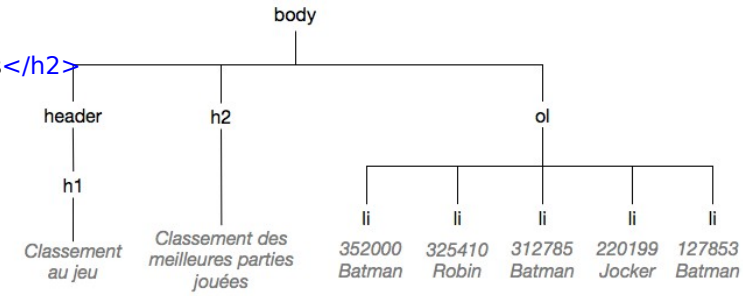
Génération automatique de pages HTML

Découpage structure vs. contenu

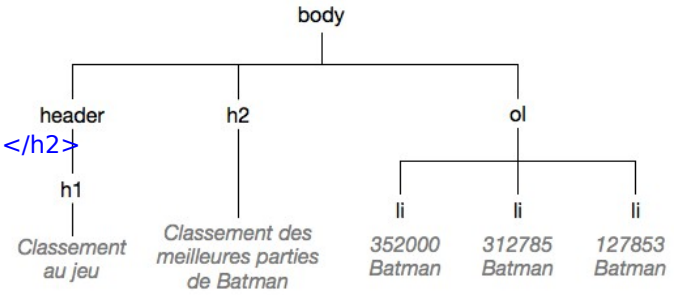
Génération automatique de pages HTML

Découpage structure vs. contenu


```
...
<body>
      <header>
    <h1> Classement au jeu </h1>
      </header>
  <h2>Classement des meilleures parties jouées</h2>
  <ol>
        <li>352100  Batman<\li>
        <li>325410  Robin<\li>
        <li>312785  Batman<\li>
        <li>220199  Jocker<\li>
        <li>127853  Batman<\li>
      </ol>
</body>
...
```

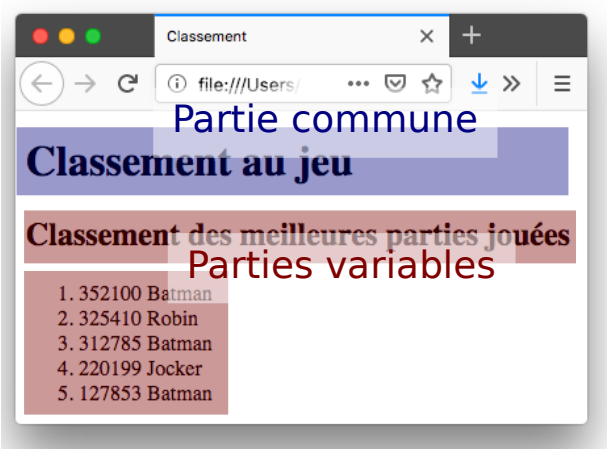



```
...
<body>
      <header>
    <h1> Classement au jeu </h1>
      </header>
  <h2>Classement des meilleures parties de Batman</h2>
  <ol>
        <li>352100  Batman<\li>
        <li>312785  Batman<\li>
        <li>127853  Batman<\li>
      </ol>
</body>
...
```

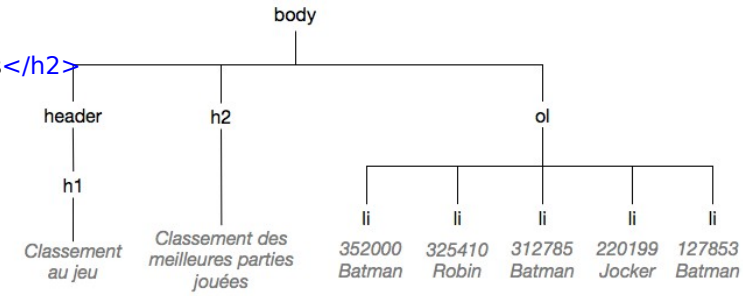


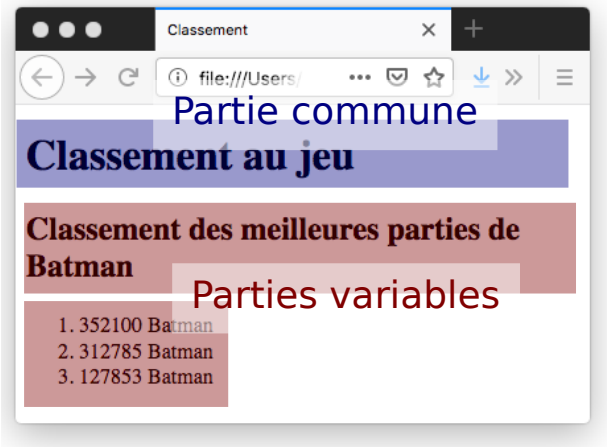
Génération automatique de pages HTML

Découpage structure vs. contenu

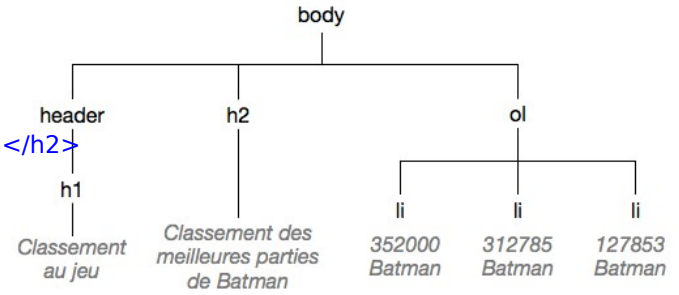



```
...
<body>
      <header>
    <h1> Classement au jeu </h1>
      </header>
  <h2>Classement des meilleures parties jouées</h2>
  <ol>
        <li>352100  Batman<\li>
        <li>325410  Robin<\li>
        <li>312785  Batman<\li>
        <li>220199  Jocker<\li>
        <li>127853  Batman<\li>
  </ol>
</body>
...
```



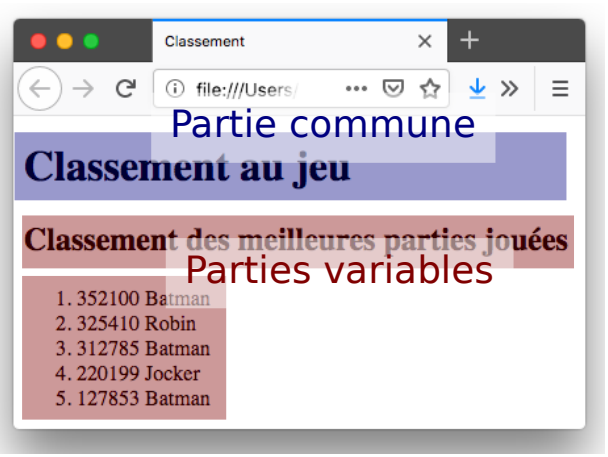



```
...
<body>
      <header>
    <h1> Classement au jeu </h1>
      </header>
  <h2>Classement des meilleures parties de Batman</h2>
  <ol>
        <li>352100  Batman<\li>
        <li>312785  Batman<\li>
        <li>127853  Batman<\li>
      </ol>
</body>
...
```

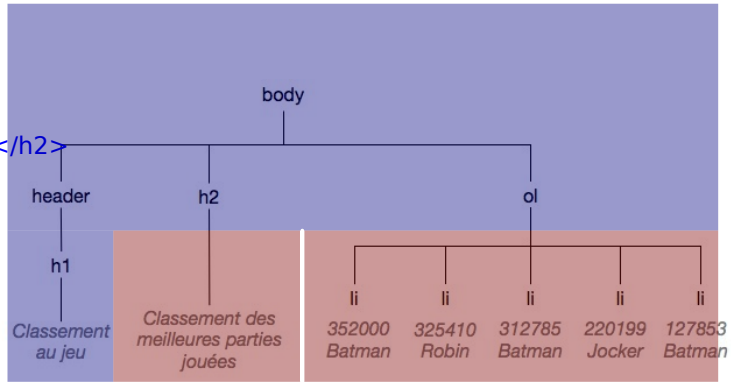


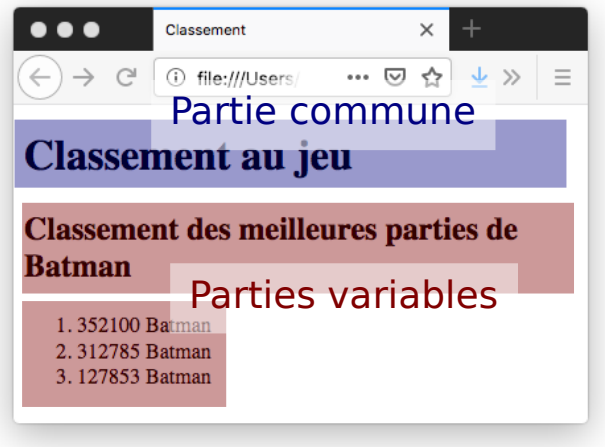
Génération automatique de pages HTML

Découpage structure vs. contenu

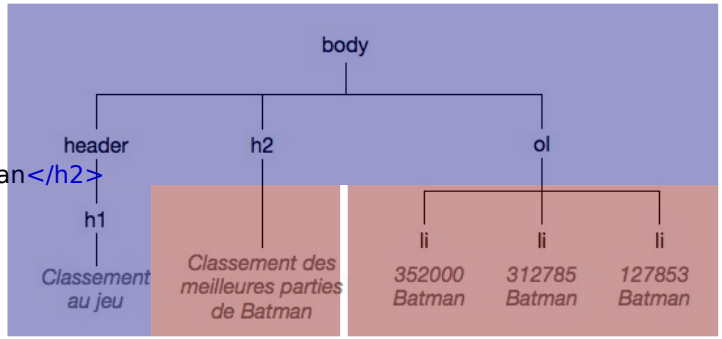



```
...
<body>
      <header>
    <h1> Classement au jeu </h1>
      </header>
  <h2>Classement des meilleures parties jouées</h2>
  <ol>
      <li>352100  Batman<\li>
      <li>325410  Robin<\li>
      <li>312785  Batman<\li>
      <li>220199  Jocker<\li>
      <li>127853  Batman<\li>
  </ol>
</body>
...
```






```
...
<body>
      <header>
    <h1> Classement au jeu </h1>
      </header>
  <h2>Classement des meilleures parties de Batman</h2>
  <ol>
      <li>352100  Batman<\li>
      <li>312785  Batman<\li>
      <li>127853  Batman<\li>
  </ol>
</body>
...
```



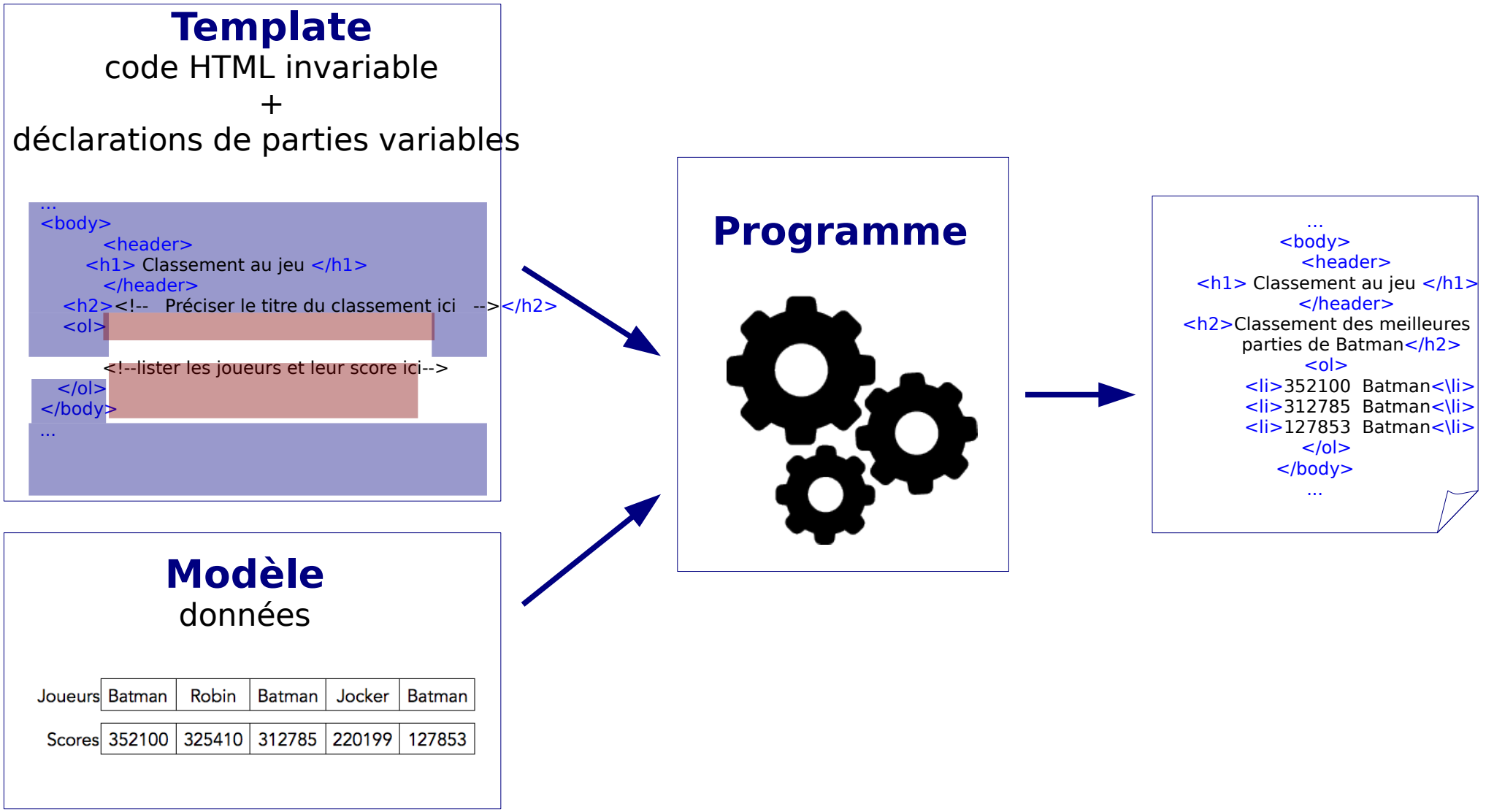
Génération automatique de pages HTML

Principe du *templating*

- Définir les données dans un fichier séparé → *Modèle*
- Définir la « partie commune » (code HTML invariable) dans un fichier HTML → *Template*
  - y déclarer des parties variables
  - On utilisera une syntaxe spéciale de *template Jinja*
- Ecrire un programme capable de générer un nouveau fichier HTML à partir d'un template et d'un modèle

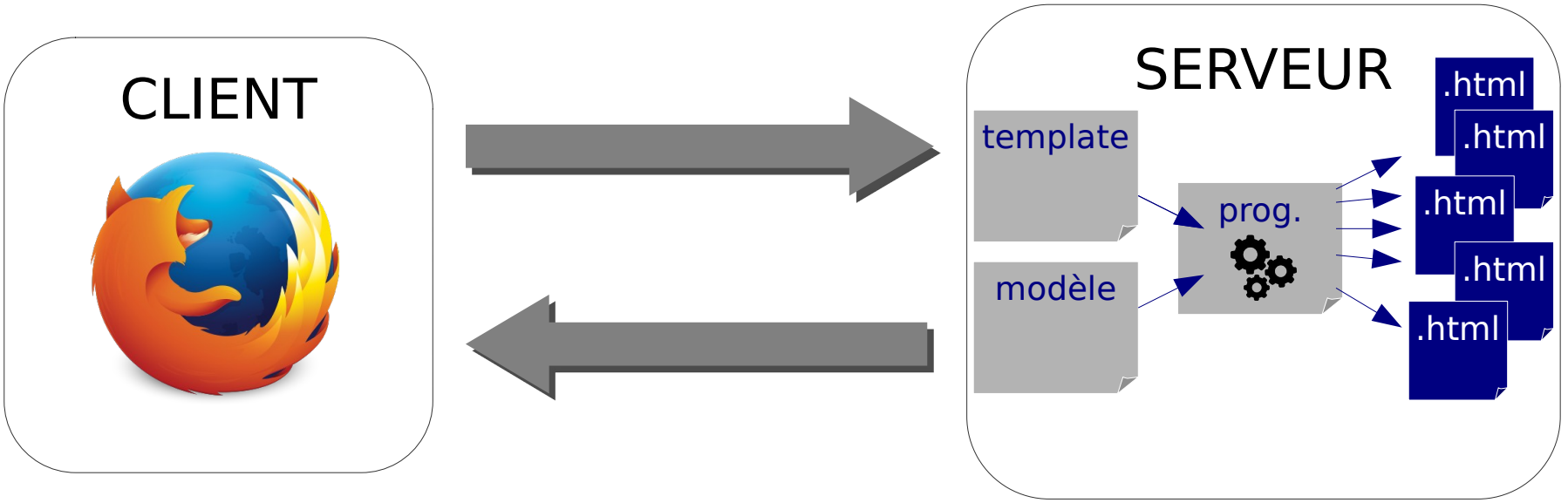
Génération automatique de pages HTML

Principe général pour générer un fichier HTML



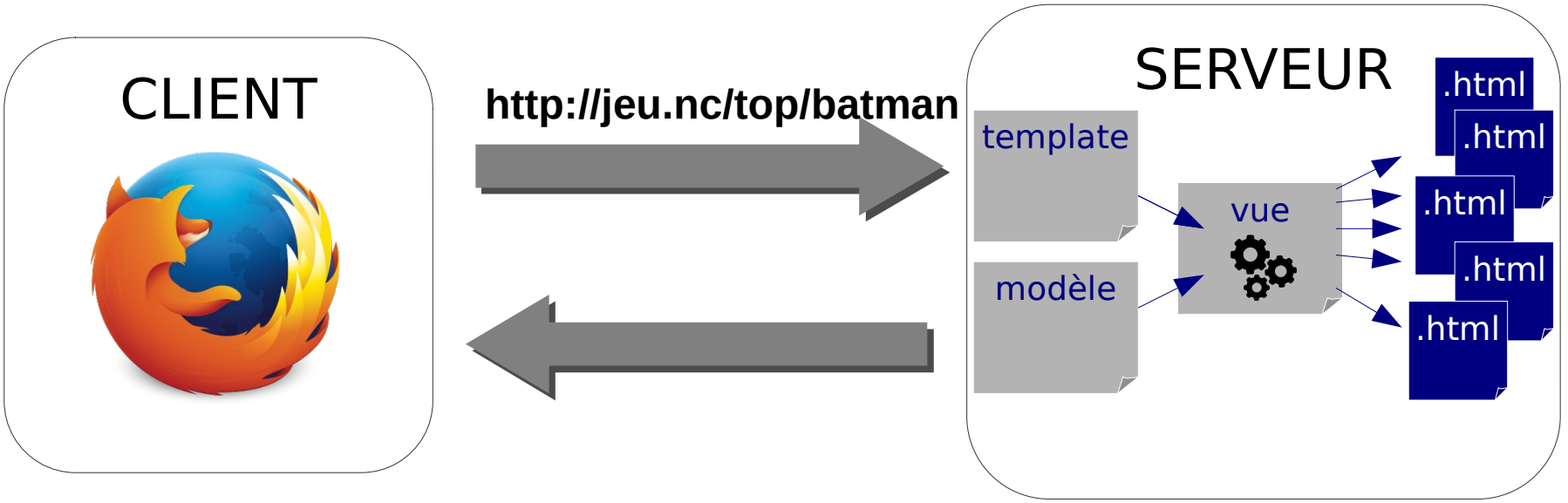
Génération automatique de pages HTML

Dans l'environnement client/serveur



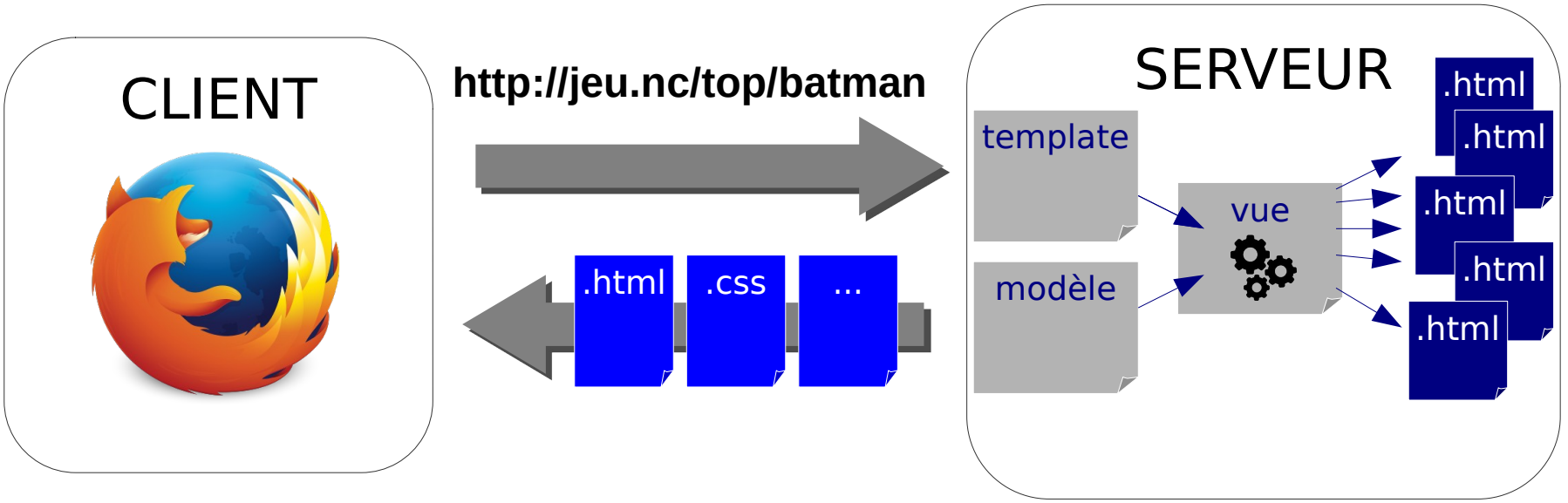
Génération automatique de pages HTML

Dans l'environnement client/serveur



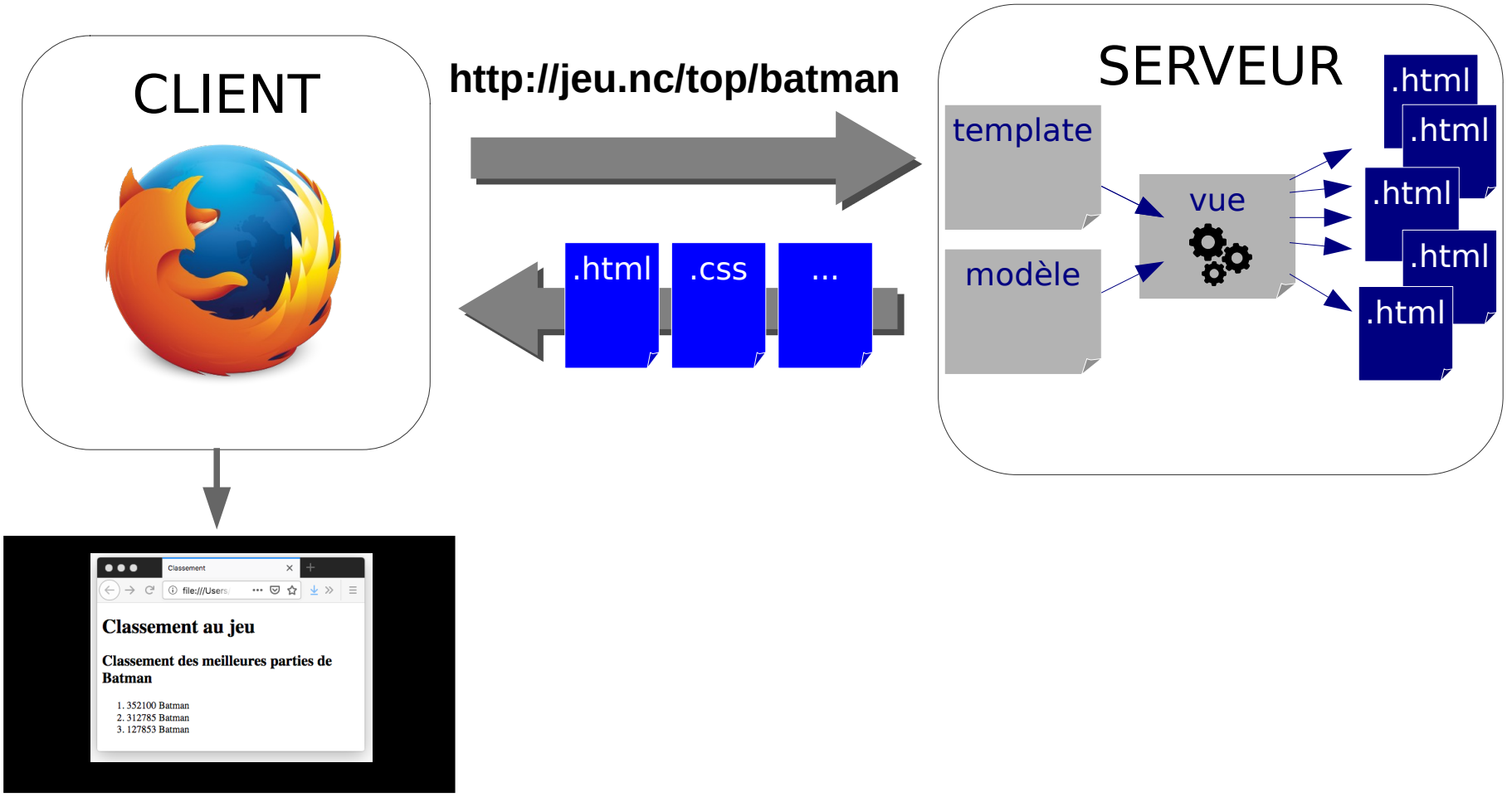
Génération automatique de pages HTML

Dans l'environnement client/serveur



Génération automatique de pages HTML

Dans l'environnement client/serveur



Génération automatique de pages HTML

En python avec la moteur de templates Jinja2

Template

.html contenant du code Python dans des balises spéciales

```
...
<body>
    <header>
  <h1> Classement au jeu </h1>
    </header>
 <h2>                    {{titre}}
 <ol>
      {% for i in range(joueurs|length) %}
       <li>{{scores[i]}} {{joueurs[i]}}<\li>
      {% endfor %}
 </ol>
</body>
...
```

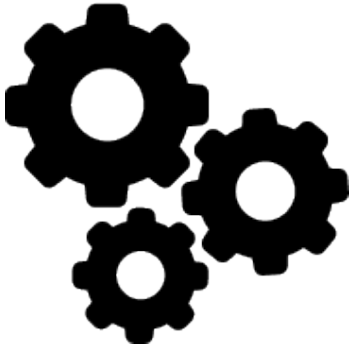
Modèle

données

Joueurs	Batman	Robin	Batman	Jocker	Batman
Scores	352100	325410	312785	220199	127853

</h2>

Programme



```
...
<body>
  <header>
<h1> Classement au jeu </h1>
  </header>
<h2>Classement des meilleures
    parties de Batman</h2>
  <ol>
<li>352100  Batman<\li>
<li>312785  Batman<\li>
<li>127853  Batman<\li>
  </ol>
</body>
...
```

Génération automatique de pages HTML

En python avec la moteur de templates Jinja2

Template

.html contenant du code Python dans des balises spéciales

```
...
<body>
    <header>
  <h1> Classement au jeu </h1>
    </header>
 <h2>                    {{titre}}
 <ol>
      {% for i in range(joueurs|length) %}
       <li>{{scores[i]}} {{joueurs[i]}}<\li>
      {% endfor %}
 </ol>
</body>
...
```

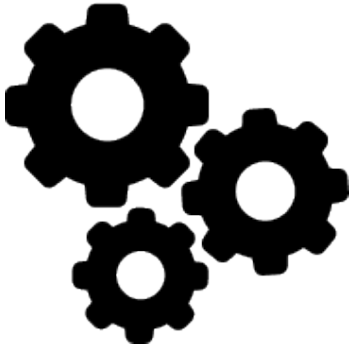
Modèle

données

Joueurs	Batman	Robin	Batman	Jocker	Batman
Scores	352100	325410	312785	220199	127853

</h2>

Programme



```
...
<body>
    <header>
  <h1> Classement au jeu </h1>
    </header>
 <h2>Classement des meilleures
     parties de Batman</h2>
    <ol>
  <li>352100  Batman<\li>
  <li>312785  Batman<\li>
  <li>127853  Batman<\li>
    </ol>
</body>
...
```

Une boucle « for » dans la syntaxe Jinja qui va permettre d'insérer un élément html <li>...<\li> par joueur

Génération automatique de pages HTML

En python avec la moteur de templates Jinja2

Template

.html contenant du code Python dans des balises spéciales

```
...
<body>
        <header>
      <h1> Classement au jeu </h1>
        </header>
   <h2>                          {{titre}}
   <ol>
        {% for i in range(joueurs|length) %}
         <li>{{scores[i]}} {{joueurs[i]}}<\li>
        {% endfor %}
   </ol>
</body>
...
```

Modèle

Script python (.py) chargeant/générant les données (depuis une BD)

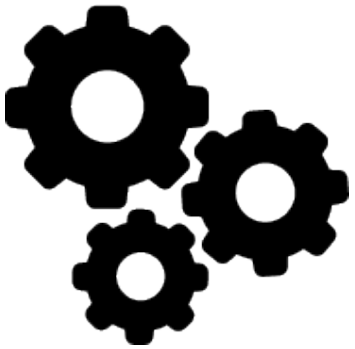
```
scores=[352100,325410,312785,220199,127853]

joueurs=['Batman','Robin','Batman','Jocker','Batman']

def scoreMoyen(scores):
   res=0
   for x in scores:
      res+=x
   return x/len(scores)
```

</h2>

Programme



```
...
<body>
        <header>
      <h1> Classement au jeu </h1>
        </header>
   <h2>Classement des meilleures
        parties de Batman</h2>
        <ol>
      <li>352100  Batman<\li>
      <li>312785  Batman<\li>
      <li>127853  Batman<\li>
        </ol>
   </body>
...
```

Génération automatique de pages HTML

En python avec la moteur de templates Jinja2

Template

.html contenant du code Python dans des balises spéciales

```
...
<body>
        <header>
      <h1> Classement au jeu </h1>
        </header>
   <h2>                          {{titre}}
   <ol>
        {% for i in range(joueurs|length) %}
         <li>{{scores[i]}} {{joueurs[i]}}<\li>
        {% endfor %}
   </ol>
</body>
...
```

Modèle

Script python (.py) chargeant/générant les données (depuis une BD)

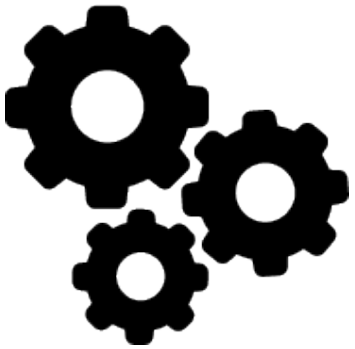
```
scores=[352100,325410,312785,220199,127853]

joueurs=['Batman','Robin','Batman','Jocker','Batman']

def scoreMoyen(scores):
   res=0
   for x in scores:
      res+=x
   return x/len(scores)
```

</h2>

Programme



```
...
<body>
   <header>
<h1> Classement au jeu </h1>
   </header>
<h2>Classement des meilleures
      parties de Batman</h2>
   <ol>
<li>352100  Batman<\li>
<li>312785  Batman<\li>
<li>127853  Batman<\li>
   </ol>
</body>
...
```

Structures de données python (listes, tuples, variables, etc.)

Fonctions python permettant de réaliser des traitements utiles sur les données (tri, calculs, filtres, etc.)

Génération automatique de pages HTML

En python avec la moteur de templates Jinja2

Template

.html contenant du code Python dans des balises spéciales

```
...
<body>
        <header>
      <h1> Classement au jeu </h1>
        </header>
   <h2>                          {{titre}}
   <ol>
      {% for i in range(joueurs|length) %}
       <li>{{scores[i]}} {{joueurs[i]}}<\li>
      {% endfor %}
   </ol>
</body>
...
```

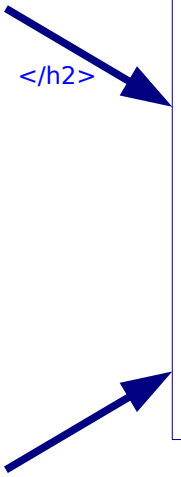
Modèle

Script python (.py) chargeant/générant les données (depuis une BD)

```
scores=[352100,325410,312785,220199,127853]

joueurs=['Batman','Robin','Batman','Jocker','Batman']

def scoreMoyen(scores):
   res=0
   for x in scores:
      res+=x
   return x/len(scores)
```



Programme

Script python (.py)

```
from jinja2 import Environment,
FileSystemLoader

from exemple_modele import *

def
creer_html(fichier_template,fichier_sortie,**inf
os):
...

creer_html("exemple_template.html",
"exemple.html",
          titre='Classement des meilleures
parties
jouées',scores=scores,joueurs=joueurs,)
```

Génération automatique de pages HTML

En python avec la moteur de templates Jinja2

exemple_template.html

Template

.html contenant du code Python dans des balises spéciales

```
...
<body>
        <header>
      <h1> Classement au jeu </h1>
        </header>
   <h2>                          {{titre}}
   <ol>
      {% for i in range(joueurs|length) %}
       <li>{{scores[i]}} {{joueurs[i]}}<\li>
      {% endfor %}
   </ol>
</body>
...
```

exemple_generateur.py

Programme

```
from jinja2 import Environment,
FileSystemLoader

from exemple_modele import *

def
creer_html(fichier_template,fichier_sortie,**inf
os):
...

creer_html("exemple_template.html",
"exemple.html",
        titre='Classement des meilleures
parties
jouées',scores=scores,joueurs=joueurs,)
```

Import du modèle (données)

Modèle

Script python (.py) chargeant/générant les données (depuis une BD)

```
scores=[352100,325410,312785,220199,127853]

joueurs=['Batman','Robin','Batman','Jocker','Batman']

def scoreMoyen(scores):
   res=0
   for x in scores:
      res+=x
   return x/len(scores)
```

exemple_modele.py

Génération automatique de pages HTML

En python avec la moteur de templates Jinja2

exemple_template.html

Template

.html contenant du code Python dans des balises spéciales

```
...
<body>
        <header>
      <h1> Classement au jeu </h1>
        </header>
   <h2>                          {{titre}}
   <ol>
        {% for i in range(joueurs|length) %}
         <li>{{scores[i]}} {{joueurs[i]}}<\li>
        {% endfor %}
   </ol>
</body>
...
```

exemple_generateur.py

Programme

```
from jinja2 import Environment,
FileSystemLoader

from exemple_modele import *

def
creer_html(fichier_template,fichier_sortie,**inf
os):
...

creer_html("exemple_template.html",
"exemple.html",
        titre='Classement des meilleures
parties
jouées',scores=scores,joueurs=joueurs,)
```

exemple.html

Modèle

Script python (.py) chargeant/générant les données (depuis une BD)

```
scores=[352100,325410,312785,220199,127853]

joueurs=['Batman','Robin','Batman','Jocker','Batman']

def scoreMoyen(scores):
   res=0
   for x in scores:
      res+=x
   return x/len(scores)
```

exemple_modele.py

Définition d'une fonction *essentielle* creer_html() :

Entrées :

- le nom du fichier template
- le nom du fichier .html à construire
- les données (du modèle) utiles

→ génère le fichier .html

Génération automatique de pages HTML

En python avec la moteur de templates Jinja2

exemple_template.html

Template

.html contenant du code Python dans des balises spéciales

```
...
<body>
        <header>
      <h1> Classement au jeu </h1>
        </header>
   <h2>                          {{titre}}
   <ol>
          {% for i in range(joueurs|length) %}
           <li>{{scores[i]}} {{joueurs[i]}}<\li>
          {% endfor %}
   </ol>
</body>
...
```

exemple_generateur.py

Programme

```
from jinja2 import Environment,
FileSystemLoader

from exemple_modele import *

def
creer_html(fichier_template,fichier_sortie,**inf
os):
...

creer_html("exemple_template.html",
"exemple.html",
        titre='Classement des meilleures
parties
jouées',scores=scores,joueurs=joueurs,)
```

exemple.html

```
...
<body>
  <header>
<h1> Classement au jeu </h1>
  </header>
<h2>
                  </h2>
  <ol>



  </ol>
</body>
...
```

Modèle

Script python (.py) chargeant/générant les données (depuis une BD)

```
scores=[352100,325410,312785,220199,127853]

joueurs=['Batman','Robin','Batman','Jocker','Batman']

def scoreMoyen(scores):
   res=0
   for x in scores:
      res+=x
   return x/len(scores)
```

exemple_modele.py

Un appel de la fonction creer_html() :

- lit le fichier exemple_template.html
- remplace les déclaration jinja par les paramètres correspondant
- exécute les structures de contrôle Jinja (ex. boucle for)

→ génère le fichier exemple.html

Génération automatique de pages HTML

En python avec la moteur de templates Jinja2

exemple_template.html

Template

.html contenant du code Python dans des balises spéciales

```
...
<body>
        <header>
      <h1> Classement au jeu </h1>
        </header>
   <h2>                          {{titre}}
   <ol>
          {% for i in range(joueurs|length) %}
           <li>{{scores[i]}} {{joueurs[i]}}<\li>
          {% endfor %}
   </ol>
</body>
...
```

exemple_generateur.py

Programme

```
from jinja2 import Environment,
FileSystemLoader

from exemple_modele import *

def
creer_html(fichier_template,fichier_sortie,**inf
os):
...

creer_html("exemple_template.html",
"exemple.html",
        titre='Classement des meilleures
parties
jouées',scores=scores,joueurs=joueurs,)
```

exemple.html

```
...
<body>
  <header>
<h1> Classement au jeu </h1>
  </header>
<h2>Classement des meilleures
    parties jouées</h2>
  <ol>


  </ol>
</body>
...
```

Modèle

Script python (.py) chargeant/générant les données (depuis une BD)

```
scores=[352100,325410,312785,220199,127853]

joueurs=['Batman','Robin','Batman','Jocker','Batman']

def scoreMoyen(scores):
   res=0
   for x in scores:
      res+=x
   return x/len(scores)
```

exemple_modele.py

Un appel de la fonction creer_html() :

- lit le fichier exemple_template.html
- remplace les déclarations jinja par les paramètres correspondant
- exécute les structures de contrôle Jinja (ex. boucle for)

→ génère le fichier exemple.html

Génération automatique de pages HTML

En python avec la moteur de templates Jinja2

exemple_template.html

Template

.html contenant du code Python dans des balises spéciales

```
...
<body>
        <header>
      <h1> Classement au jeu </h1>
        </header>
   <h2>                          {{titre}}
   <ol>
        {% for i in range(joueurs|length) %}
          <li>{{scores[i]}} {{joueurs[i]}}<\li>
        {% endfor %}
   </ol>
</body>
...
```

exemple_generateur.py

Programme

```
from jinja2 import Environment,
FileSystemLoader

from exemple_modele import *

def
creer_html(fichier_template,fichier_sortie,**inf
os):
...

creer_html("exemple_template.html",
"exemple.html",
        titre='Classement des meilleures
parties
jouées',scores=scores,joueurs=joueurs,)
```

exemple.html

```
...
<body>
  <header>
<h1> Classement au jeu </h1>
  </header>
<h2>Classement des meilleures
    parties jouées</h2>
  <ol>


  </ol>
</body>
...
```

Modèle

Script python (.py) chargeant/générant les données (depuis une BD)

```
scores=[352100,325410,312785,220199,127853]

joueurs=['Batman','Robin','Batman','Jocker','Batman']

def scoreMoyen(scores):
   res=0
   for x in scores:
      res+=x
   return x/len(scores)
```

exemple_modele.py

Un appel de la fonction creer_html() :

- lit le fichier exemple_template.html
- remplace les déclarations jinja par les paramètres correspondant
- exécute les structures de contrôle Jinja (ex. boucle for)

Génération automatique de pages HTML

En python avec la moteur de templates Jinja2

exemple_template.html

Template

fichier .html contenant du pseudo-code (proche du python)

```
...
<body>
    <header>
  <h1> Classement au jeu </h1>
    </header>
 <h2>                    {{titre}}
 <ol>
    {% for i in range(joueurs|length) %}
     <li>{{scores[i]}} {{joueurs[i]}}<\li>
    {% endfor %}
 </ol>
</body>
...
```

exemple_generateur.py

Programme

```
from jinja2 import Environment,
FileSystemLoader

from exemple_modele import *

def
creer_html(fichier_template,fichier_sortie,**inf
os):
...

creer_html("exemple_template.html",
"exemple.html",
        titre='Classement des meilleures
parties
jouées',scores=scores,joueurs=joueurs,)
```

</h2>

exemple.html

```
...
<body>
  <header>
<h1> Classement au jeu </h1>
  </header>
<h2>Classement des meilleures
    parties jouées</h2>
  <ol>


  </ol>
</body>
...
```

Modèle

Script python (.py) chargeant/générant les données (depuis une BD)

```
scores=[352100,325410,312785,220199,127853]

joueurs=['Batman','Robin','Batman','Jocker','Batman']

def scoreMoyen(scores):
   res=0
   for x in scores:
      res+=x
   return x/len(scores)
```

exemple_modele.py

Un appel de la fonction creer_html() :

- lit le fichier exemple_template.html
- remplace les déclarations jinja par les paramètres correspondant
- exécute les structures de contrôle Jinja (ex. boucle for)

Génération automatique de pages HTML

En python avec la moteur de templates Jinja2

exemple_template.html

Template

fichier .html contenant du pseudo-code (proche du python)

```
...
<body>
        <header>
      <h1> Classement au jeu </h1>
        </header>
   <h2>                          {{titre}}
   <ol>
          {% for i in range(joueurs|length) %}
           <li>{{scores[i]}} {{joueurs[i]}}<\li>
           {% endfor %}
   </ol>
</body>
...
```

exemple_generateur.py

Programme

```
from jinja2 import Environment,
FileSystemLoader

from exemple_modele import *

def
creer_html(fichier_template,fichier_sortie,**inf
os):
...

creer_html("exemple_template.html",
"exemple.html",
        titre='Classement des meilleures
parties
jouées',scores=scores,joueurs=joueurs,)
```

exemple.html

```
...
<body>
  <header>
<h1> Classement au jeu </h1>
  </header>
<h2>Classement des meilleures
    parties jouées</h2>
    <ol>
<li>352100  Batman<\li>
<li>312785  Batman<\li>
<li>127853  Batman<\li>
    </ol>
</body>
...
```

Modèle

Script python (.py) chargeant/générant les données (depuis une BD)

```
scores=[352100,325410,312785,220199,127853]

joueurs=['Batman','Robin','Batman','Jocker','Batman']

def scoreMoyen(scores):
   res=0
   for x in scores:
      res+=x
   return x/len(scores)
```

exemple_modele.py

Un appel de la fonction creer_html() :

- lit le fichier exemple_template.html
- remplace les déclarations jinja par les paramètres correspondant
- exécute les structures de contrôle Jinja (ex. boucle for)

Démonstration galerie d'images

- Télécharger le contenu de https://insight.nc/
  - Extraire les adresses des fond d'écrans
  - Extraire les villes et pays
- Créer un modèle
  - template HTML
  - CSS type « galerie » avec flexbox
- Générer un fichier HTML avec modèle + template

Le moteur de templating Jinja

https://jinja.palletsprojects.com/en/3.0.x/

Quelques éléments de syntaxe Jinja

Les variables du template

Le code Jinja introduit des variables dans le template (HTML). Ces expressions seront évaluées à l'exécution de la vue, elles seront remplacées par leur valeur lors de la génération de la page HTML finale.

Template

```
…
<body>
  <h1> {{prenom}} {{nom}} </h1>
  <p>{{prenom}} {{nom}} a été
     président(e) du gouvernement de
     Nouvelle Calédonie.</p>
</body>
…
```

Programme

```
...

creer_html('presidents_template.html',
           'fichier.html',
           nom='Durand',
           prenom='Bernard')
...
```

fichier.html

```
…
<body>
  <h1> Bernard Durand </h1>
  <p>Bernard Durand a été
     président(e) du gouvernement de
     Nouvelle Calédonie.</p>
</body>
…
```

Exécution de la vue
Le **rendering**

Quelques éléments de syntaxe Jinja

Les variables du template

Le code Jinja introduit des variables dans le template (HTML). Ces expressions seront évaluées à l'exécution de la vue, elles seront remplacées par leur valeur lors de la génération de la page HTML finale.

Une variable Jinja se note entre doubles accolades : `{{ expression }}` lorsqu'elle est à remplacer dans la page HTML à générer.

Elle **doit être définie** par les paramètres utilisés lors de l'appel du template.

Quelques éléments de syntaxe Jinja

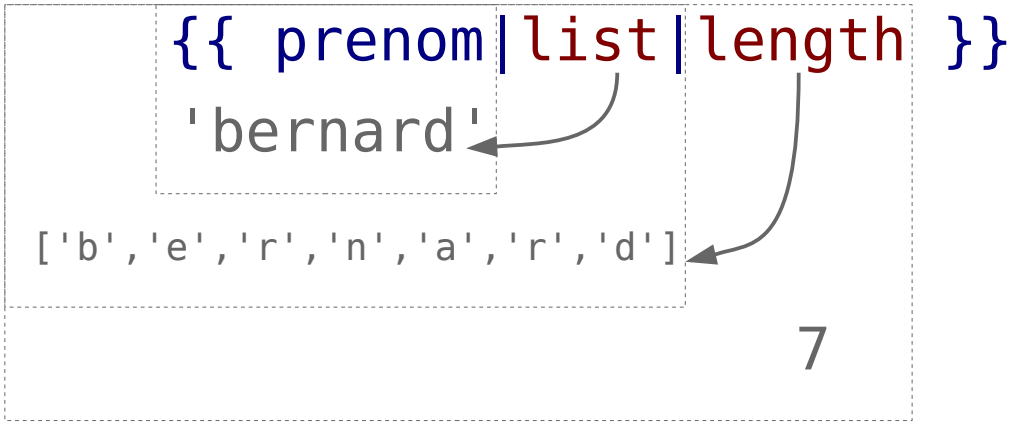
Les filtres Jinja

Les filtres doivent être vus comme des fonctions prédéfinies dans Jinja et qui permettent de modifier les variables du template.

L'application d'un `filtre` sur une `variable` se note `{{ variable | filtre}}`

Plusieurs filtres peuvent être « chaînés », chaque filtre est alors appliqué sur la sortie du précédent.

Exemple :



Quelques éléments de syntaxe Jinja

Les filtres Jinja

Les filtres doivent être vus comme des fonctions prédéfinies dans Jinja et qui permettent de modifier les variables du template.

L'application d'un `filtre` sur une `variable` se note `{{ variable | filtre}}`

Plusieurs filtres peuvent être « chaînés », chaque filtre est alors appliqué sur la sortie du précédent.

Une 50aine de filtres Jinja dont :

- join : concatène les éléments d'une liste dans une chaîne de caractères
- length : taille d'une séquence (liste, tuple, etc.)
- sum : somme d'une séquence de nombres
- ...

Quelques éléments de syntaxe Jinja

Les tests Jinja

Jinja préféfinit des (fonctions de) tests afin de « tester » certaines propriétés sur des variables.

L'application d'un test `prop` sur une `variable` se note `variable` **`is`** `prop` et retourne `VRAI` ou `FAUX` selon que `variable` satisfait ou non au test `prop`.

Une 20aine de tests Jinja dont :

- defined, undefined : vérifie qu'une variable a bien été définie dans le dictionnaire des paramètres
- none : vérifie qu'une variable est `None`
- number, string : vérifie qu'une variable est un nombre/un string
- ...

Quelques éléments de syntaxe Jinja

Les structures de contrôle Jinja

Comme tout langage de programmation, Jinja offre des structures permettant de contrôler algorithmiquement le contenu de la page générée.

Ces structures de contrôles se déclarent par `{% … %}`

Conditionnelle

```
{% if condition %}
{# instructions #}
{% elif condition %}
{# instructions #}
{% else %}
{# instructions #}
{% endif %}
```

Quelques éléments de syntaxe Jinja

Les structures de contrôle Jinja

Comme tout langage de programmation, Jinja offre des structures permettant de contrôler algorithmiquement le contenu de la page générée.

Ces structures de contrôles se déclarent par `{% … %}`

Conditionnelle

```
{% if condition %}
{# instructions #}
{% elif condition %}
{# instructions #}
{% else %}
{# instructions #}
{% endif %}
```

facultatifs

Quelques éléments de syntaxe Jinja

Les structures de contrôle Jinja

Comme tout langage de programmation, Jinja offre des structures permettant de contrôler algorithmiquement le contenu de la page générée.

Ces structures de contrôles se déclarent par `{% … %}`

Conditionnelle

```
{% if condition %}
{# instructions #}
{% elif condition %}
{# instructions #}
{% else %}
{# instructions #}
{% endif %}
```

Boucle for

```
{% for elem in liste %}
{# instructions #}
{% endfor %}

{% for i in range(liste|length) %}
{# instructions #}
{% endfor %}
```

Quelques éléments de syntaxe Jinja

Les structures de contrôle Jinja

Comme tout langage de programmation, Jinja offre des structures permettant de contrôler algorithmiquement le contenu de la page générée.

Ces structures de contrôles se déclarent par `{% … %}`

Boucle for

Parcours d'une liste par ses éléments

```
{% for elem in liste %}
# instructions #}
% endfor %}
```

Parcours d'une liste par ses indices

```
{% for i in range(liste|length) %}
# instructions #}
% endfor %}
```

Quelques éléments de syntaxe Jinja

Les structures de contrôle Jinja

Comme tout langage de programmation, Jinja offre des structures permettant de contrôler algorithmiquement le contenu de la page générée.

Ces structures de contrôles se déclarent par `{% … %}`

Exemple :

```
<ol>
  {% for i in range(sequence|length) %}
    {% if sequence[i] is string %}
      <li value='{{i}}' class= 'string'>{{sequence[i]}}</li>
    {% elif sequence[i] is number %}
      <li value='{{i}}' class= 'number'>{{sequence[i]}}</li>
    {% endif %}
  {% endfor %}
</ol>
```

Quelques éléments de syntaxe Jinja

Les structures de contrôle Jinja

Comme tout langage de programmation, Jinja offre des structures permettant de contrôler algorithmiquement le contenu de la page générée.

Ces structures de contrôles se déclarent par {% ...

Exercice

Exemple :

```
<ol>
  {% for i in range(sequence|length) %}
    {% if sequence[i] is string %}
      <li value='{{i}}' class= 'string'>{{sequence[i]}}</li>
    {% elif sequence[i] is number %}
      <li value='{{i}}' class= 'number'>{{sequence[i]}}</li>
    {% endif %}
  {% endfor %}
</ol>
```

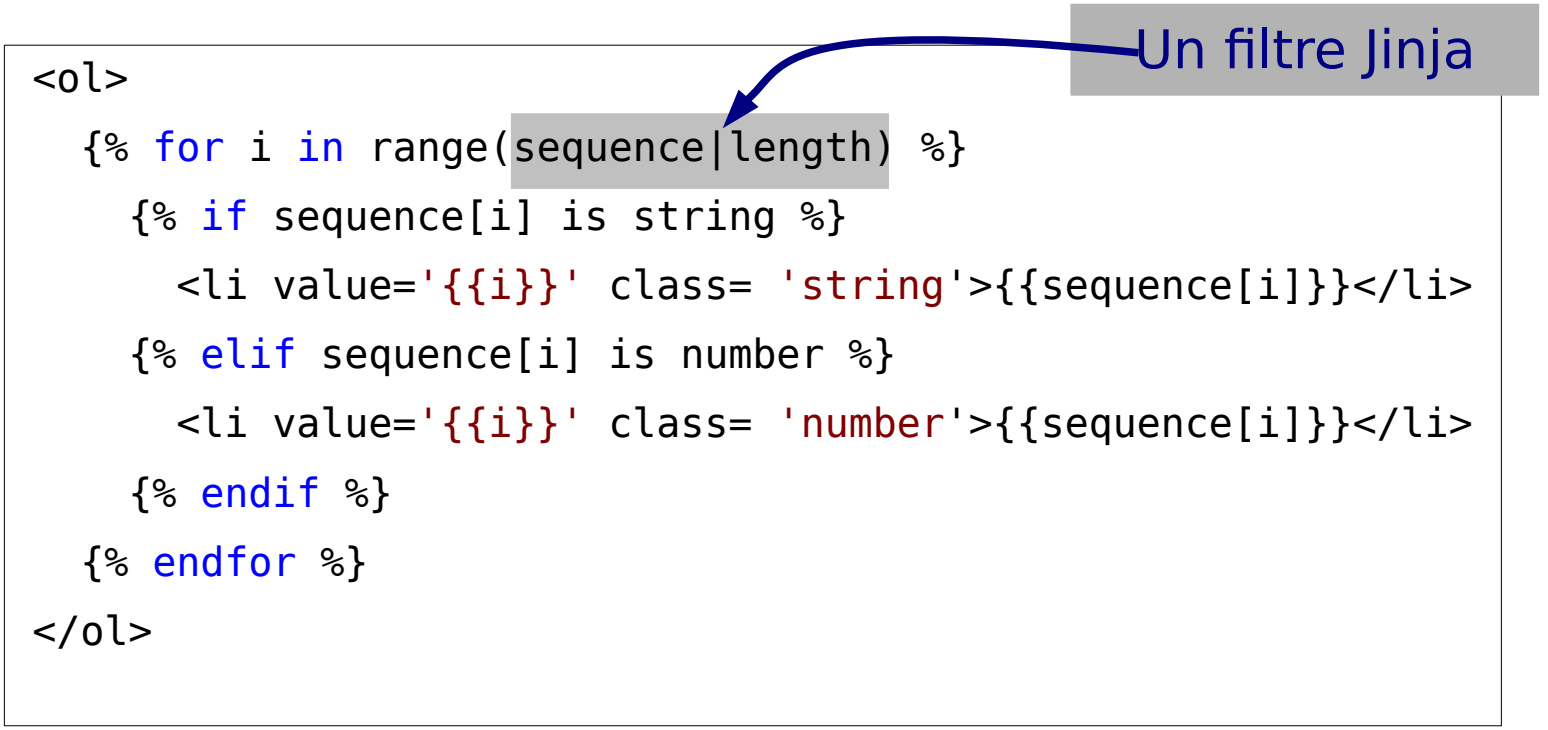
Quelques éléments de syntaxe Jinja

Les structures de contrôle Jinja

Comme tout langage de programmation, Jinja offre des structures permettant de contrôler algorithmiquement le contenu de la page générée.

Ces structures de contrôles se déclarent par {% … %}

Exemple :



Quelques éléments de syntaxe Jinja

Les structures de contrôle Jinja

Comme tout langage de programmation, Jinja offre des structures permettant de contrôler algorithmiquement le contenu de la page générée.

Ces structures de contrôles se déclarent par {% … %}

Exemple :

des tests Jinja

```
<ol>
  {% for i in range(sequence|length) %}
    {% if sequence[i] is string %}
      <li value='{{i}}' class= 'string'>{{sequence[i]}}</li>
    {% elif sequence[i] is number %}
      <li value='{{i}}' class= 'number'>{{sequence[i]}}</li>
    {% endif %}
  {% endfor %}
</ol>
```

Quelques éléments de syntaxe Jinja

Les structures de contrôle Jinja

Comme tout langage de programmation, Jinja offre des structures permettant de contrôler algorithmiquement le contenu de la page générée.

Ces structures de contrôles se déclarent par {% … %}

Exemple :

Affichages : {{ }} → print

```
<ol>
  {% for i in range(sequence|length) %}
    {% if sequence[i] is string %}
      <li value='{{i}}' class= 'string'>{{sequence[i]}}</li>
    {% elif sequence[i] is number %}
      <li value='{{i}}' class= 'number'>{{sequence[i]}}</li>
    {% endif %}
  {% endfor %}
</ol>
```

Quelques éléments de syntaxe Jinja

Les structures de contrôle Jinja

Comme tout langage de programmation, Jinja offre des structures permettant de contrôler algorithmiquement le contenu de la page générée.

Ces structures de contrôles se déclarent par {% … %}

Exemple :

```
<ol>
  {% for i in range(sequence|length) %}
    {% if sequence[i] is string %}
      <li value='{{i}}' class= 'string'>{{sequence[i]}}</li>
    {% elif sequence[i] is number %}
      <li value='{{i}}' class= 'number'>{{sequence[i]}}</li>
    {% endif %}
  {% endfor %}
</ol>
```

Utilisation des variables sans affichage (pas de {{ }})

Quelques éléments de syntaxe Jinja

Les structures de contrôle Jinja

Comme tout langage de programmation, Jinja offre des structures permettant de contrôler algorithmiquement le contenu de la page générée.

- Algorithmes (Jinja) dans le template → décider **comment** les données sont affichées
- Algorithmes (python) dans le modèle → décider **quelles** données sont affichées

Questions ?