

Etude de cas : PARSER XML/HTML

A) Qu'est-ce que le XML/HTML?

a. Définition du XML

XML signifie eXtensible Markup Language : en français, c'est un langage de balisage extensible (markup=balise)
XML est devenu un format d'échange de données incontournable aussi bien que le format CSV.

Le **XML** permet la mise en forme de documents via l'utilisation de balises. Développé et standardisé par le World Wide Web Consortium (W3C) à la fin des années 1990, il répondait à l'objectif de définition d'un langage simple, facile à mettre en application.

Le **XML** se classe dans la catégorie des langages de description (il n'est ni un langage de programmation, ni un langage de requêtes). Il est donc naturellement utilisé pour décrire des données en s'appuyant sur des balises et des règles personnalisables.

- C'est un langage de balisage :
 - Cela signifie qu'on accole aux données des « étiquettes » qui qualifient leur contenu (qu'on appelle une balise. Une balise se repère avec des délimiteurs : **<** et **>**
- C'est un langage extensible, cela signifie deux choses en fait :
 - selon ses besoins, on peut définir et utiliser les « étiquettes » que l'on souhaite ;
 - une fois qu'un jeu d'étiquettes est défini, même par quelqu'un d'autre, on peut l'étendre en y ajoutant ses propres créations.

Il ne faut en fait pas parler de langage XML au singulier, mais bien de langages au pluriel. En effet, XML désigne un certain type de fichier texte, respectant des règles d'écriture.

b. Structure d'un fichier XML

Fichier exemple :

```
<?xml version="1.0" encoding="UTF-8"?>
<savants>
  <!-- Quelques physiciens importants -->
  <savant id="phys0">
    <identité prénom="Galileo" nom="Galilei"/>
    <dates naissance="1564-02-15" décès="1642-01-08"/>
    <contributions>Relativité du mouvement & amp; système
    héliocentrique</contributions>
  </savant>
  <savant id="phys1">
    <identité prénom="Isaac" nom="Newton"/>
    <dates naissance="1643-01-04" décès="1727-03-31"/>
    <contributions>Gravitation universelle, principe d'inertie, décomposition de la
    lumière & amp; calcul infinitésimal</contributions>
  </savant>
  <savant id="phys2">
    <identité nom="Einstein" prénom="Albert"/>
    <dates naissance="1879-03-14" décès="1955-04-18"/>
    <contributions><![CDATA[Relativités restreinte & générale, nature corpusculaire de
    la lumière & effet photoélectrique]]></contributions>
  </savant>
</savants>
```

➤ Le prologue

```
<?xml version="1.0" encoding="UTF-8"?>
```

Cette première ligne est le *prologue* du fichier. Il indique la version de la recommandation XML qui sert de base à l'écriture du fichier (il n'y a eu à cette date qu'une seule version, la version 1.0), ainsi que l'encodage de caractères utilisé. Cette ligne est facultative.

➤ Les commentaires

```
<!-- Quelques physiciens importants -->
```

Un commentaire, comme en HTML, commence par la chaîne de caractères <!-- et se termine par -->. Tout ce qui est entre ces deux chaînes de caractères n'est pas une donnée du document, mais un commentaire laissé par le développeur

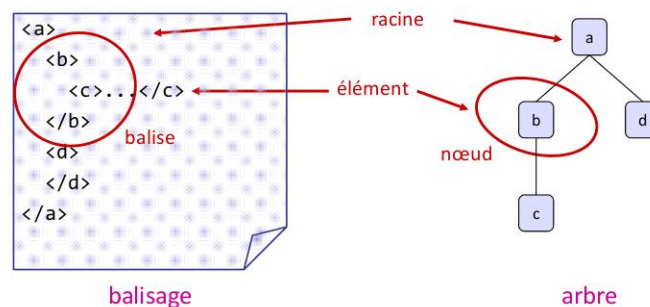
➤ Les éléments et attributs

Nous abordons là les étiquettes dont nous avons parlé en guise d'introduction. Ce sont les éléments et attributs qui permettent de qualifier les données, et de leur conférer un sens.

✓ Les éléments

Il existe deux types d'éléments : les éléments vides... et ceux qui ne le sont pas.

- **Un élément non vide commence par une balise ouvrante et se termine par une balise fermante.** Dans le document précédent, des exemples de balises ouvrantes sont <savants> ou <contributions> ; les balises fermantes correspondantes sont respectivement </savants> et </contributions>.
- **Un élément vide ne comporte qu'une seule balise, dite « auto-fermante ».** C'est le cas de l'élément identité : il est caractérisé par la balise <identité... : vous noterez qu'elle se termine par />.



L'inclusion des balises les unes dans les autres crée un lien de parenté (les balises parents ont des balises enfant dans leur intérieur). On peut représenter cette hiérarchie de parenté avec un arbre. Cette structure hiérarchique avec des nœuds et des contenus d'appelle le **DOM** (Document Object Model : modèle de document objet)

✓ Les attributs

- **Quand on veut préciser ce que contient un élément, on peut placer un attribut sur la balise ouvrante.** Par exemple, l'élément date comporte deux attributs, nommés naissance et décès. Un attribut possède un nom, ainsi qu'une valeur. Par exemple, l'attribut prénom du deuxième savant a pour valeur Isaac.
- **Les attributs ne sont jamais repris dans la balise fermante :** l'attribut id présent dans la balise ouvrante de l'élément savant est absent de la balise fermante </savant>.
- Enfin, l'ordre dans lequel les attributs sont écrits dans une balise ouvrante n'a aucune importance : les attributs du dernier élément savant sont inversés par rapport aux deux premiers, mais les deux formes sont strictement équivalentes.

➤ Les entités

Vous aurez noté que certains caractères ont une signification en XML : c'est le cas par exemple des caractères < ou >. Quand on doit les utiliser, il faut donc trouver une parade pour qu'ils ne soient pas interprétés. Pour cela, on utilise des *entités*, qui codent ces caractères par des chaînes de caractères les représentant ou des nombres.

Une entité commence par le caractère & et se termine par un point-virgule ;. Par exemple, le caractère < doit être écrit < ou <. En particulier, le caractère & doit lui-même être codé comme une entité sous la forme &

➤ Les sections CDATA

Lorsqu'un contenu est susceptible de contenir plusieurs entités, il peut être fastidieux de les écrire. On utilise alors ce que l'on appelle une « section CDATA », qui est destinée à contenir des informations affichées telles quelles, sans traitement par l'outil de consultation (navigateur Internet ou tout autre type d'application chargée de traiter le document XML).

C'est le cas dans notre exemple où l'on a écrit <![CDATA[Relativités restreinte & générale, nature corpusculaire de la lumière & effet photoélectrique]]> : il n'a pas été nécessaire de remplacer chaque occurrence de & par l'entité correspondante &

b. Définition du HTML

Le HTML (Hyper Text Markup Language) est une **norme** de codage XML particulière où la liste des balises et des attributs possibles par balise a été **décidée par le W3C** de façon fixe. Le HTML sert de langage de codage pour écrire des pages web. Il est destiné à décrire des documents Hypertexte (= texte amélioré avec mise en forme, gras, souligné, italique, couleur, images, liens, etc : ce qu'on appelle une page web)

Ici la liste complète des balises de la norme HTML : <https://www.w3schools.com/tags/default.asp>

Ici la liste des attributs de balises HTML autorisés selon les balises : https://www.w3schools.com/tags/ref_attributes.asp

Donc la façon dont on écrit un document HTML est identique à la façon dont on écrit un document XML, sauf que les balises ne peuvent pas être inventées comme on le souhaite, mais selon une norme précise. La norme actuelle est le HTML 5. La structure d'un document HTML précise aussi des balises obligatoires dans un document et leur ordre de parenté.

Pour indiquer que le document est de type HTML plutôt que XML, on conseille une 1^{ère} ligne de prologue (toujours facultative) pour les documents HTML : `<!DOCTYPE html>`

B) Qu'est-ce qu'un parser XML/HTML?

Définition :

parser = lire et analyser syntaxiquement un flux de donnée pour le découper en unité syntaxique connexes.

C'est l'action d'exploiter un fichier sous une forme brute afin d'en tirer les informations utiles. Cette action est généralement réalisée à partir d'un fichier XML ou CSV. Afin de faciliter le traitement de l'information, on a tendance à parser le fichier afin d'en soutirer les informations importantes et les insérer dans une base de données ou une structure de données de type fichier .

Un parser est donc un analyseur du contenu d'une source de données pour chercher à en comprendre le contenu et le trier et l'organiser d'une façon ordonnée.

Concrètement, un parser est un ensemble de sous-programmes spécifiques à développer (gestionnaires ou handler), qui s'appuient sur une bibliothèque que l'on trouve dans les différents environnements de programmation (Visual studio, PHP, Python, ...).

Le but ici dans notre problème est de parcourir un fichier texte au format XML ou HTML contenant des balises, des données et des attributs ayant des valeurs, et donc de pouvoir récupérer les données et les valeurs dans leur contexte (balise).

Dans le cadre de l'étude de cas, il s'agit d'alimenter les traits (vecteurs de traits) à partir de fichiers contenant le code source de pages web. Le format de sortie est fixé par un format XML épuré, les valeurs binaires (positive, negative) seront déduites du nombre d'étoiles (3 minimum pour positive)

Ici on fait un parser XML/HTML car on va récupérer des données à analyser provenant d'un fichier HTML (site web), qui est un cas particulier de format XML. Les données analysées et récupérées sont ensuite stockées et organisées dans un fichier avec un certain format choisi. Ici on choisit de représenter les données stockées aussi au format XML. Donc on analyse un HTML (cas de norme XML particulière) contenant plein de données et on produit un XML contenant des données analysées, classées, triées de façon ciblée.

C) Principe d'un parser XML et HTML

a. Où peuvent être les données qui nous intéressent pour le parser?

Certaines balises permettent de structurer et ne contiennent aucune donnée.

Dans l'exemple suivant issu d'un relevé GPS :

Nous pouvons constater qu'un **Trackpoint** est composé d'une date/heure, d'une position, d'une altitude...

Une position est composée d'une latitude et d'une longitude.

Les valeurs effectives que nous devons rechercher peuvent se retrouver à 2 endroits.

```
<Trackpoint>
  <Time>2006-05-14T06:10:38Z</Time>
  <Position>
    <LatitudeDegrees>43.129576</LatitudeDegrees>
    <LongitudeDegrees>2.558775</LongitudeDegrees>
  </Position>
  <AltitudeMeters>201.052856</AltitudeMeters>
  <DistanceMeters>0.000000</DistanceMeters>
  <HeartRateBpm>110</HeartRateBpm>
  <Cadence>21</Cadence>
  <SensorState>Absent</SensorState>
</Trackpoint>
```

b. Valeur d'attribut

Dans l'exemple suivant :

```
<Lap StartTime="2006-05-14T06:10:38Z">
  ...
</Lap>
```

Nous avons la balise `Lap` qui contient l'attribut `StartTime` qui possède la valeur :

`2006-05-14T06:10:38Z`

c. Valeur entre balises

Dans l'exemple suivant : `<Cadence>21</Cadence>`

La balise `Cadence` contient la valeur 21.

D) Programmation à mettre en place en PHP pour un Parser HTML

Il existe un ensemble de fonctions PHP permettant de parser des fichiers XML, de façon native. Mais comme le langage HTML est moins strict que le XML, le chargement d'une page HTML dans un parser XML risque d'échouer. Nous allons donc utiliser une bibliothèque qui a été développée pour cela : simple HTML DOM.

Il faut donc copier le fichier `simple_html_dom.php` dans l'espace de votre application et en faire un `include`.

ATTENTION : cette bibliothèque fonctionne très bien en PHP 5 mais pas en PHP 7, veuillez à votre version de PHP

Comme il s'agit d'une bibliothèque orientée objet, il faut créer une instance de la classe `simple_html_dom`. Et ensuite lier le fichier à parser avec la fonction (non objet) `file_get_html` :

```
file_get_html(localisation relative du fichier);
```

Vous pouvez aussi créer un parser à partir d'une chaîne de caractères contenant du code HTML avec :

```
str_get_html(untextehtml);
```

Donc le code avec l'objet nommé `$html` sera :

```
$html= simple_html_dom();
$html= file_get_html (localisation relative du fichier);
```

Maintenant nous pouvons récupérer une collection contenant tous les éléments html qui correspondent à la même balise :

```
$macollection_input = $html->find('input');
```

Grâce à la boucle `foreach` nous pouvons la parcourir :

```
foreach($macollection_input as $un_element){
  ...
}
```

Vous pouvez aussi parcourir de manière classique :

```
$nb=count($macollection);
for($i=0; $i<$nb;$i++){
  ...
}
```

Lorsque vous voulez cibler un élément par son id vous pouvez procéder ainsi :

```
$macollection = $html->getElementById('unid');
```

L'élément de la collection étant lui-même un objet, il sera doté d'un tableau `attr` qui permettra de récupérer la valeur de chaque attribut:

```
$un_element -> lenomdunattribut
```

```
ou: $un_element -> attr['lenomdunattribut']
```

Vous aurez probablement besoin de tester l'existence d'un attribut car certaines balises de la même famille peuvent avoir certains attributs et d'autres non.

Vous disposez de la fonction PHP `isset`:

```
isset(unevariable) renvoie true si la variable existe ou false sinon.
```

Et donc pour savoir si un élément possède ou non un attribut :

```
isset($un_element -> lenomdunattribut)
```

```
ou : isset($un_element -> attr['lenomdunattribut'])
```

Et enfin si vous voulez accéder à ce qui est contenu entre une balise ouvrante et fermante :

```
$unelement->nodes[...] ou $unelement->innertext ou $unelement->plaintext
```

innertext et *plaintext* ne sont pas équivalents dans la façon dont ils récupèrent les données

nodes permet de récupérer les nœuds du DOM contenus dans la balise, c'est-à-dire les sous-contenus des balises situées dedans ; sous la forme d'un tableau.

innertext fait un affichage de tous les *nodes* de l'élément les uns à la suite des autres.

Donc *innertext* correspond à afficher tous les *nodes* d'un coup.

```
// Exemple : Texte HTML à analyser
```

```
$html = str_get_html("<div>foo <b>bar</b></div>");
```

```
// Renvoie le tableau des éléments ou la valeur "null"
```

```
// si balise non trouvée (avec le zéro en 2ème argument)
```

```
$e = $html->find("div", 0);
```

```
echo $e->tag; // Renvoie: " div"
```

```
echo $e->outertext; // Renvoie: " <div>foo <b>bar</b></div>"
```

```
echo $e->innertext; // Renvoie: " foo <b>bar</b>"
```

```
echo $e->plaintext; // Renvoie: " foo bar"
```

```
/*
```

```
Nom d'attribut      Utilisation
```

```
$e->tag              Lit ou écrit le nom de balise d'un élément
```

```
$e->outertext        Lit ou écrit le contenu HTML texte extérieur de l'élément.
```

```
$e->innertext        Lit ou écrit le contenu HTML texte intérieur de l'élément.
```

```
$e->plaintext        Lit ou écrit le contenu texte brut de l'élément.
```

```
*/
```

E) Compléments pour la programmation PHP

a) JSON

JSON : format de stockage plus récent utilisant des systèmes de parenthésage permettant de délimiter les objets et les descripteurs sont séparés par des virgules avec des couples :
" descripteur " = "valeur "

Définition

JSON : JavaScript Object Notation

JSON est un format de données textuelles dérivé de la notation des objets du langage JavaScript.

Il permet de représenter de l'information structurée comme le permet XML.

Créé par Douglas Crockford entre 2002 et 2005, il est décrit par la RFC 7159 de l'IETF.

En JSON, objet et descripteurs se retrouvent sous les formes suivantes :

- Un objet est un ensemble non ordonné de paires **identificateur : valeur**
Un objet commence avec **{** et se termine avec **}**
Les noms sont suivis de **:** et les paires **identificateur: valeur** sont séparées par des **,**
- Un tableau est une collection ordonnée de valeurs.
Un tableau commence avec **[** et se termine avec **]**.
Les valeurs sont séparées par des **,**
- Une valeur peut être une chaîne de caractères entourées par des guillemets doubles, un nombre, un booléen, la valeur null, un objet ou un tableau.

Exemple de données au format JSON :

```

1  {
2      "prenom": "Pierre",
3      "nom": "Giraud",
4      "adresse": {
5          "rue": "30 Impasse des Lilas",
6          "ville": "Toulon",
7          "cp": 83000,
8          "pays": "France"
9      },
10     "mails": [
11         "pierre.giraud@edhec.com",
12         "pierre@pierre-giraud.com"
13     ]
14 }
```

Lorsqu'un attribut possède une valeur au format JSON :

`{ "champ1": "valeur1", "champ2": "valeur2", "champ3": valeur3, ... }`

Vous pouvez récupérer les couples identificateurs/valeurs soit en utilisant la fonction `explode` vue en cours, soit utiliser la fonction `json_decode` :

```
$ch='{ "champ1": "valeur1", "champ2": "valeur2", "champ3": 10 }';
$t=json_decode($ch);
```

Et vous pouvez accéder aux différentes valeurs de la manière suivante :

```
$t->champ1 : donnera le texte valeur1
$t->champ3 : donnera la valeur 10
```

b) Parcours de dossier

Pour parcourir un dossier, il faut un objet qui représente le dossier cible:

`$dossier = opendir(cheminrelatifdudossiercible);`**Notes :** `opendir` — Ouvre un dossier, et récupère un pointeur dessus`opendir (string $path [, resource $context]) : resource|false`

`opendir()` retourne un pointeur sur un dossier qui peut être utilisé avec les fonctions `closedir()`, `readdir()` et `rewinddir()`

Ensuite on pourra faire une boucle de parcours :

```
while($element_dossier = readdir($dossier)){
    ...
}
```

Notes : `readdir` — Lit une entrée du dossier`readdir ([resource $dir_handle]) : string|false``readdir()` retourne le nom de la prochaine entrée du dossier identifiée par `dir_handle`.

Les entrées sont retournées dans l'ordre dans lequel elles sont enregistrées dans le système de fichiers.

Par contre chaque élément du dossier peut être un fichier ou un sous-dossier.

La fonction `is_dir(unélément)` donne `true` si l'élément est un *dossier*, `false` si c'est un *fichier*.

c) Ecriture dans un fichier

Il faut d'abord créer un fichier en mode écriture :

```
$fic = fopen(localiationrelativeetnomdufichier, 'w');
```

Attention: le dossier cible doit exister !

Pour écrire une ligne dans le fichier :

```
fputs($fic, letexteàécrire);
```

Note: Si vous voulez provoquer un retour à la ligne dans le fichier, il faut concaténer `"\n"` à la fin de la ligne (attention aux guillemets)

Manuel en ligne de simple dom HTML :

<http://simplehtmldom.sourceforge.net/manual.htm>

Vous pourrez y trouvez des compléments vous permettant de produire un code plus concis...

ACTIVITES préparatoires en PHP

Activité 1

Ecrire le programme PHP qui permet d'obtenir l'affichage suivant en extrayant les données depuis le fichier [page_exo1.html](#) grâce à l'objet principal de la bibliothèque simple HTML DOM :

Indications :

- Analyser le contenu du fichier de la page HTML avec un objet simple HTML DOM
- Parcourir l'ensemble des balises `<h1>` (pour récupérer le contenu de l'unique balise `h1`) et en afficher le contenu
- Parcourir l'ensemble des balises `<td>` et en afficher le contenu

Rappel : lorsqu'on récupère le contenu d'une balise, qu'il y en ait une ou plusieurs dans le document, c'est un objet de type tableau associatif PHP qui est récupéré. S'il y a un seul élément à prendre, le tableau associatif aura donc une seule case, la première qu'on prend.

Premier titre

v_1_1
v_1_2
v_2_1
v_2_2
v_3_1
v_3_2

Activité 2

Ecrire le programme PHP qui permet d'obtenir l'affichage suivant en extrayant les données depuis le fichier [page_exo1.html](#) grâce à l'objet principal de la bibliothèque simple HTML DOM :

Le but est de faire le même affichage que précédemment trois fois, mais avec des méthodes différentes : (et en ajoutant un affichage texte en PHP de la méthode utilisée avant chaque affichage)

- ❖ une fois avec : `$unelement->innertext`
- ❖ une fois avec : `$unelement->plaintext`
- ❖ une fois avec : `$unelement->nodes [...]`

Premier titre

Avec innertext

v_1_1v_1_2
v_2_1v_2_2
v_3_1v_3_2

Avec plaintext

v_1_1v_1_2
v_2_1v_2_2
v_3_1v_3_2

Avec nodes

v_1_1v_1_2
v_2_1v_2_2
v_3_1v_3_2

Activité 3

Ecrire le programme PHP qui permet d'obtenir l'affichage suivant en extrayant les données depuis le fichier [page_exo1.html](#) grâce à l'objet principal de la bibliothèque simple HTML DOM :

Le but ici est de récupérer chaque ligne du contenu du tableau (chaque contenu de balise `<tr>`) et de traiter chacune de ces **lignes de texte HTML** (contenant chacune des balises HTML `<td>`) par un autre objet simple HTML DOM afin d'en afficher le contenu (le contenu des balises `<td>`).

Comme vous le constatez on fait un affichage PHP supplémentaire (« Cellule 1 », « Cellule 2 ») qui permet de faire une étiquette d'information des éléments affichés, qui sont numérotés (avec une variable de comptage de cellules en PHP)

Premier titre

Cellule 1 : v_1_1 Cellule 2 : v_1_2

Cellule 1 : v_2_1 Cellule 2 : v_2_2

Cellule 1 : v_3_1 Cellule 2 : v_3_2

Rappel : pour alimenter un objet simple HTML DOM par une **ligne de texte HTML** :

```
str_get_html(untextehtml);
```

On pourra aussi faire une version bis qui parcourt les lignes `<tr>` du tableau avec une boucle `for` classique.

Activité 4

Ecrire le programme PHP qui permet d'obtenir l'affichage suivant en extrayant les données depuis le fichier [page_exo2.html](#) grâce à l'objet principal de la bibliothèque simple HTML DOM :

Indications :

- Analyser le contenu du fichier de la page HTML avec un objet simple HTML DOM
- Parcourir l'ensemble des balises <table> (tableaux en HTML)
- Pour chaque <table> trouvé, afficher une information **Tableau i** (+ retour à la ligne)
- Puis analyser le contenu texte HTML de chaque balise <table> avec un objet simple HTML DOM
- Parcourir l'ensemble des balises <tr> (lignes du tableau)
- Pour chaque <tr> trouvé, afficher une information **ligne i** (+ retour à la ligne)
- Puis analyser le contenu texte HTML de chaque balise <tr> avec un objet simple HTML DOM
- Parcourir l'ensemble des balises <td> (cellules de la ligne du tableau)
- Afficher le contenu de la cellule (en le faisant précéder de l'affichage d'une information **Cellule i**) (+ retour à la ligne après l'ensemble des contenus des cellules du <td>)

```
Tableau 1
ligne 1
Cellule 1 : v_1_1 Cellule 2 : v_1_2
ligne 2
Cellule 1 : v_2_1 Cellule 2 : v_2_2
ligne 3
Cellule 1 : v_3_1 Cellule 2 : v_3_2
Tableau 2
ligne 1
Cellule 1 : v2_1_1 Cellule 2 : v2_1_2
ligne 2
Cellule 1 : v2_2_1 Cellule 2 : v2_2_2
ligne 3
Cellule 1 : v2_3_1 Cellule 2 : v2_3_2
```

On pourra aussi faire une version bis utilisant le fichier [page_exo2bis.html](#) qui affiche les titres <h1> des tableaux

Activité 5

Ecrire le programme PHP qui permet d'obtenir l'affichage suivant en extrayant les données depuis le fichier [page_exo3.html](#) grâce à l'objet principal de la bibliothèque simple HTML DOM :

On cherche non pas à afficher le contenu de tous les tableaux de la page HTML, mais à afficher seulement le tableau qui a l'identifiant « tbl2 » du fichier HTML. On « cible » donc la balise cherchée par son id

```
Ciblage du tableau d'id tbl2
ligne 1
Cellule 1 : v2_1_1 Cellule 2 : v2_1_2
ligne 2
Cellule 1 : v2_2_1 Cellule 2 : v2_2_2
ligne 3
Cellule 1 : v2_3_1 Cellule 2 : v2_3_2
```

Ensuite on adopte la même stratégie pour afficher le contenu qu'à l'activité précédente.

Rappel : pour cibler une balise par son id:

```
$macollection = $html->getElementById('unid');
```

Activité 6

Ecrire le programme PHP qui permet d'obtenir l'affichage suivant en extrayant les données depuis le fichier [page_exo4.html](#) grâce à l'objet principal de la bibliothèque simple HTML DOM :

Le but est ici de faire afficher le contenu des attributs de certaines balises.

Dans cet exemple on n'utilise que des balises situées dans l'en-tête du document, le <head>, mais on pourrait très bien faire exactement la même chose pour des balises situées dans le <body> ayant des attributs !

```
Contenu (attribut content) : text/html; charset=utf-8
Contenu (attribut content) : IUT_Carcassonne
Nom (attribut name) : author
```

Rappel 1: Si on a une collection de tous les éléments HTML qui correspondent à la même balise:

```
$macollection_input = $html->find('nomdebalise');
```

Alors on peut accéder à un attribut d'une variable élément d'une collection:

```
$un_element -> attr['lenomdunattribut']
```

ou: `$un_element -> lenomdunattribut`

Avec: `$un_element` est une des « cases » du tableau associatif `$macollection_input`

Et: « `lenomdunattribut` » est un attribut de la balise « `nomdebalise` »

Rappel 2: Pour savoir si un élément possède ou non un attribut :

```
isset($un_element -> attr['lenomdunattribut'])
```

ou: `isset($un_element -> lenomdunattribut)`

→ renvoie True si l'attribut existe

Activité 7

Ecrire le programme PHP qui permet d'obtenir l'affichage suivant en extrayant les données depuis le fichier `page_exo5.html` grâce à l'objet principal de la bibliothèque simple HTML DOM :

On récupère le contenu JSON de l'attribut 'data' :

```
{ "Nom": "PROF", "Prénom": "Informatique", "Taille": 1.78 }
```

Et on affiche le contenu de deux façons différentes :

- ❖ Avec : `json_decode`
- ❖ Avec : `explode`

Version JSON
Nom: PROF
Prénom: Informatique
Taille: 1.78

Version explode
Nom PROF
Prénom Informatique
Taille 1.78

Concernant la fonction PHP `explode` : Scinde une chaîne de caractères en segments

`explode(separateur, string, limit)`

`explode` (**string** \$delimiter, **string** \$string [, **int** \$limit = **PHP_INT_MAX**]) : **array**

`explode()` retourne un tableau de chaînes de caractères, chacune d'elle étant une sous-chaîne du paramètre string extraite en utilisant le séparateur delimiter.

// Exemple 1

```
$pizza = "piece1 piece2 piece3 piece4 piece5 piece6";
$pieces = explode(" ", $pizza);
echo $pieces[0]; // piece1
echo $pieces[1]; // piece2
```

// Exemple 2

```
$chaine = 'one|two|three|four';
print_r(explode('|', $chaine));
//Affiche: Array ( [0] => one [1] => two [2] => three [3] => four )
```

Activité 8

Ecrire le programme PHP qui permet d'obtenir l'affichage suivant (parcours du dossier `rep_captures`):

On parcourt l'ensemble des fichiers et dossiers contenus au premier niveau dans le dossier « `rep_captures` »

Rappel :

Pour parcourir un dossier, il faut un objet qui représente le dossier cible:

```
$dossier = opendir(cheminrelatifdudossiercible);
```

Ensuite on pourra faire une boucle de parcours :

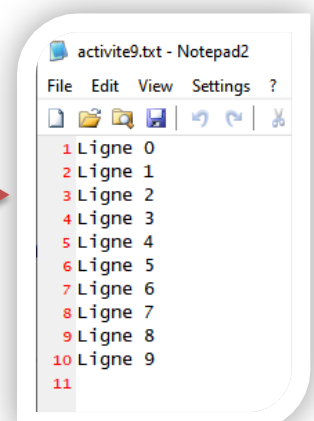
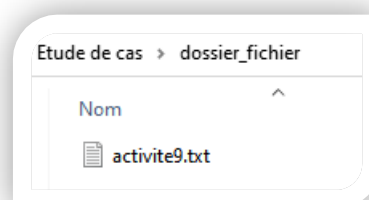
```
while($element_dossier = readdir($dossier)){
    ...
}
```

Dossier : .
Dossier : ..
Fichier : serie001_p1.html
Fichier : serie001_p2.html
Fichier : serie001_p3.html
Fichier : serie002_p1.html
Fichier : serie003_p1.html
Fichier : sous_dossier

Activité 9

Ecrire le programme PHP qui permet de créer un fichier « `activite9.txt` » texte stocké dans le sous-dossier « `dossier_fichier` » et qui contiendra 10 lignes de la forme :

Ligne xx où xx sera une valeur allant de 0 à 9
(à alimenter par une boucle !)



Construction du parser PHP par étapes successives

Activité 10 : Parser v1 (on parse un seul fichier)

Ecrire le programme PHP qui parse le fichier « `serie001_p1.html` » du dossier « `rep_capture` » (contenant quelques fichiers de capture AlloCine de 2017) et qui crée dans le sous-dossier « `rep_sorties_xml` » le fichier « `corpusserie001_p1.xml` ».

Le fichier XML devra respecter la forme suivante :

```
<?xml version="1.0" encoding="utf-8" ?>
<CORPUS>
<SERIE_COMMENTAIRES>
<SERIE id="11303" >Peaky Blinders</SERIE>
<COMMENTAIRES>
<COMMENTAIRE id="44578762" auth="Silly Man" eval="5">
  Bonjour ! Pour ma première critique...
</COMMENTAIRE>
<COMMENTAIRE ...>
  ...
</COMMENTAIRE>
  ...
</COMMENTAIRES>
</SERIE_COMMENTAIRES>
</CORPUS>
```

On a donc besoin des informations suivantes à extraire depuis le document source HTML:

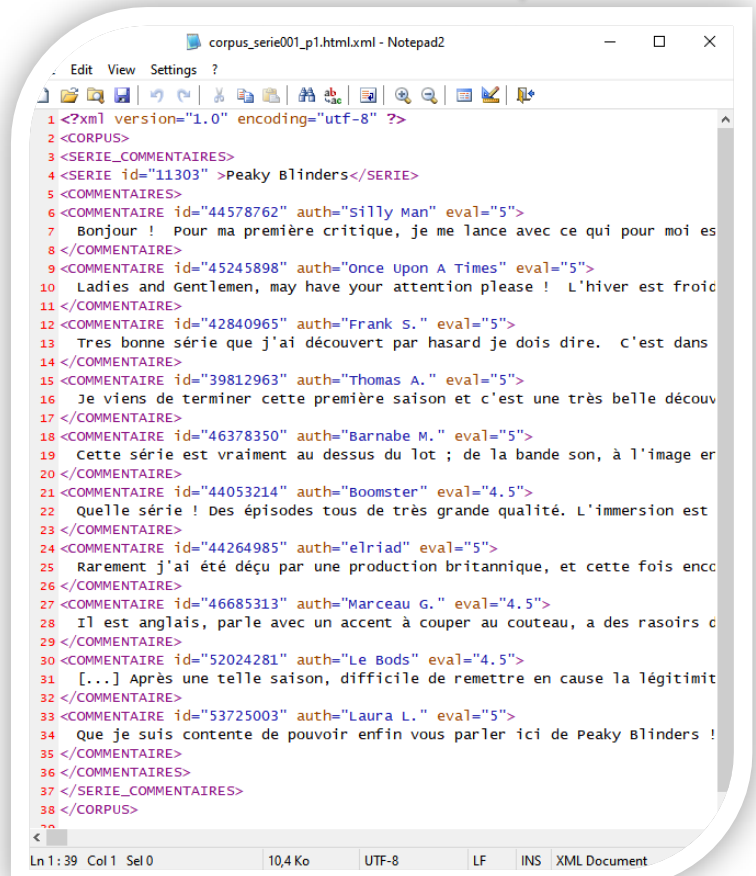
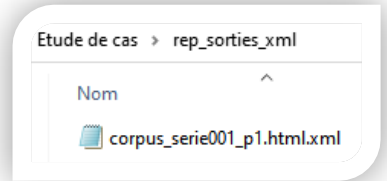
- ✓ Code de la série : **11303**
- ✓ Titre de la série : **Peaky Blinders**
- ✓ Code du commentaire : **44578762**
- ✓ Pseudo de l'auteur : **Silly Man**
- ✓ Evaluation laissée par l'auteur notée sur 5 : **5**
- ✓ Commentaire en lui-même : **Bonjour ! Pour ma première critique...**

Indications :

Pour cela il faut bien sûr d'abord passer du temps à observer la façon dont sont structurées et encapsulées les données qui nous intéressent dans le fichier source « `serie001_p1.html` ».

- Cherchez où est le nom de la série (son titre), regardez bien les balises `<meta>` → extrayez-le
- Cherchez où est le n° code numérique de la série, regardez bien les balises `<meta>` → extrayez-le
- Quelle balise encapsule systématiquement chacun des commentaires ? Si d'autres contenus du document qui ne sont pas des commentaires apparaissent aussi dans le même type de balise, cherchez quels attributs distinguent les balises spécifiques aux commentaires [voir : `itemprop`]
- Dans quel attribut de la balise contenant le commentaire est encodé au format JSON le code du commentaire, pseudo de l'auteur et évaluation laissée ? Décodez-le avec la fonction `json_decode` pour récupérer ces informations.
- ATTENTION : pour le commentaire en lui-même, remarquez que parfois dans la balise qui le contient, il est suivi d'autres balises `` ou autre avec des liens et ce qu'on veut c'est le commentaire seulement, pas ce qui suit. → A récupérer donc avec `nodes[0]` (premier contenu HTML de la balise qui contient le commentaire, avant toute autre balise qui suivrait dedans)

Votre script PHP pourra aussi sortir à l'affichage écran les informations récupérées sous un format propre du genre de celui donné ci-dessous, même si le but principal est de constituer le fichier « `corpusserie001_p1.xml` ».



Code série: 11303 - Titre: Peaky Blinders

Auteur: Silly Man - Note: 5 - Code commentaire: 44578762

Bonjour ! Pour ma première critique, je me lance avec ce qui pour moi est un chef d'oeuvre aboslu, The Peaky Blinders. C'est simple je n'y vois aucun défaut, les acteurs sont tous exceptionnels, le lieu de tournage et le décor sont non seulement immersifs mais aussi d'une beauté rare (voir jamais vu) à la télévision, sans parler du scénario. Pour incarner le personnage principal, Thomas Shelby les producteurs ont choisi, Cillian Murphy, surtout connu pour son rôle dans Inception, montre ici tout l'étendu de son talent, c'est simple pour moi l'Emmy Awards lui vient de droit pour ce rôle (l'année dernière à part Matthew Mcconaughey personne ne pouvait rivaliser avec Bryan Cranston aka Walter White). Il apporte une profondeur et un charisme au personnage de Thomas Shelby, à tel point qu'on en vient à admirer ce dernier. Annabelle Wallis est divine, Sam Neill est également excellent avec une présence à l'écran qui lui est propre, sans parler de tout l'entourage Shelby. À la saison 2 Tom Hardy rejoint son ex camarade d'Inception, Cillian Murphy, dans un personnage d'Alfie Solomons aussi original que remarquable ! Quant au scénario, on a une histoire plus qu'aboutie, les dialogues sont très bien fournis, de plus le spectateur est vite récompensé de sa fidélité contrairement à des séries type Mentalist ou Dexter, où on ne va pas assez vite au but. Ici en 12 épisodes réparties par 6 sur 2 saisons, on est vraiment accroché à l'histoire jusqu'au bout, on en démont pas, on aime chaque personnage même les plus "terribles". Pour le décor, nous sommes dans le Birmingham des années 20, avec des éléments qui rappelle la révolution industrielle. Les lieux sont magnifiques, vraiment des tableaux, à chaque point de vue on pourrait prendre une belle photo. C'est simple même le générique, Nick Cave - Red Right Hand, est un chef d'oeuvre. Vraiment vivement la saison 3, d'ici là je vous la recommande sans modération !

Auteur: Once Upon A Times - Note: 5 - Code commentaire: 45245898

Ladies and Gentlemen, may have your attention please ! L'hiver est froid, l'attente trop longue. Je compte désormais parmi mes séries de haute volée : Peaky Blinders. Une série non seulement bien jouée, qui repose sur une bande son rock Artic Monkeys et Nick Cave à l'honneur mais qui se dote d'un scénario béton. Oui un scénario, chose qui fait malheureusement défaut sur le paysage des séries actuelles. Une question: Pourquoi cette série n'est-elle pas plus connue ? Mystère, en attendant bon visionnage.

Affichage écran du travail réalisé en plus de la constitution du fichier (facultatif)

Activité 11 : Parser v2 (on parse plusieurs fichiers)

Ecrire le programme PHP qui parse tous les fichiers du dossier « rep_captures » et qui crée dans le sous dossier « rep_sorties_xml » le fichier « corpus.xml ».

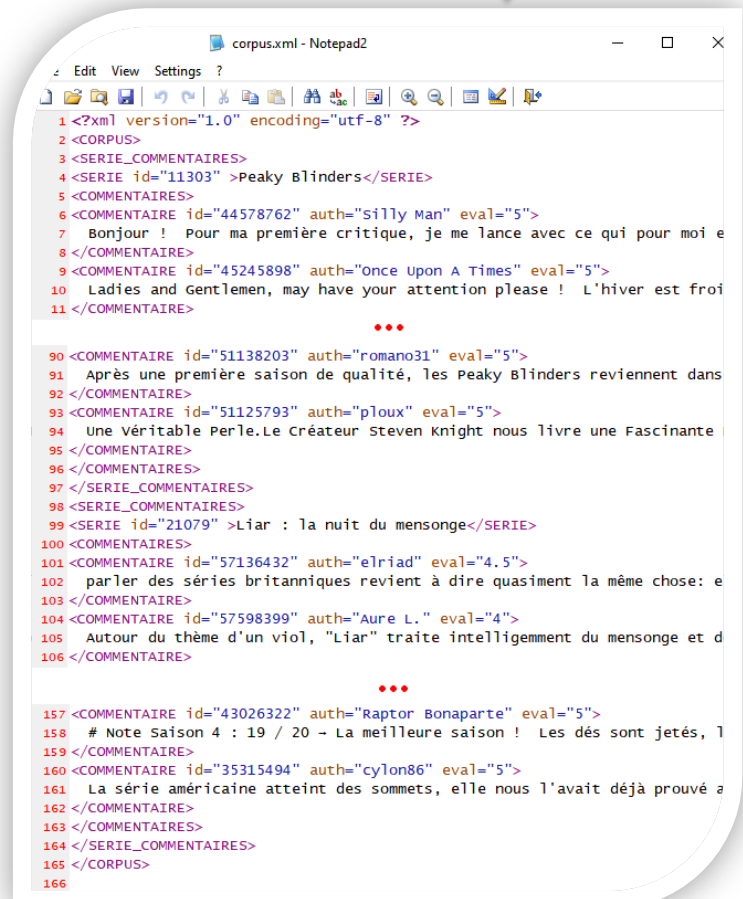
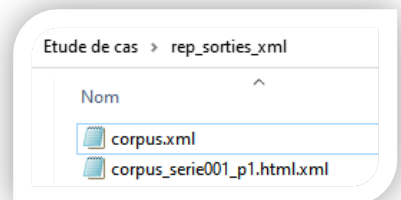
C'est donc le travail fait précédemment à l'activité 10 mais on le fait sur tous les fichiers HTML du répertoire et le résultat est mis dans un même fichier corpus.

De plus tous les commentaires concernant une même série (de même code numérique) qui sont dans des fichiers HTML source consécutifs du répertoire (il faut que la capture soit prévue ainsi), doivent être mis dans la même section <SERIE> du fichier XML, on ne crée des sections <SERIE> différentes que pour des séries différentes.

Le fichier XML devra respecter la forme suivante :

```
<?xml version="1.0" encoding="utf-8" ?>
<CORPUS>
<SERIE_COMMENTAIRES>
<SERIE id="11303" >Peaky Blinders</SERIE>
<COMMENTAIRES>
<COMMENTAIRE id="44578762" auth="Silly
Man" eval="5">
    Bonjour ! Pour ma première critique
</COMMENTAIRE>
<COMMENTAIRE ...>
    ...
</COMMENTAIRE>
    ...
</COMMENTAIRES>
</SERIE_COMMENTAIRES>

<SERIE_COMMENTAIRES>
<SERIE id="..." >...</SERIE>
<COMMENTAIRES>
<COMMENTAIRE id="..." auth="..."
eval="...">
    ...
</COMMENTAIRE>
<COMMENTAIRE ...>
    ...
</COMMENTAIRE>
    ...
</COMMENTAIRES>
</SERIE_COMMENTAIRES>
    ...
</CORPUS>
```



Affichage écran facultatif par le script PHP des informations recueillies ci-dessous :

Fichier traité: serie001_p1.html

Code série: 11303 - Titre: Peaky Blinders

Auteur: Silly Man - Note: 5 - Code commentaire: 44578762

Bonjour ! Pour ma première critique, je me lance avec ce qui pour moi est un chef d'oeuvre aboslu, The Peaky Blinders. C'est simple je n'y vois aucun défaut, les acteurs sont tous exceptionnels, le lieu de tournage et le décor sont non seulement immersifs mais aussi d'une beauté rare (voir jamais vu) à la télévision, sans parler du scénario. Pour incarner le personnage principal, Thomas Shelby les producteurs ont choisi, Cillian Murphy, surtout connu pour son rôle dans Inception, montre ici tout l'étendu de son talent, c'est simple pour moi l'Emmy Awards lui vient de droit pour ce rôle (l'année dernière à part Matthew McConaughey personne ne pouvait rivaliser avec Bryan Cranston aka Walter White). Il apporte une profondeur et un charisme au personnage de Thomas Shelby, à tel point qu'on en vient à admirer ce dernier. Annabelle Wallis est divine, Sam Neill est également excellent avec une présence à l'écran qui lui est propre, sans parler de tout l'entourage Shelby. À la saison 2 Tom Hardy rejoint son ex camarade d'Inception, Cillian Murphy, dans un personnage d'Alfie Solomons aussi original que remarquable ! Quant au scénario, on a une histoire plus qu'aboutie, les dialogues sont très bien fournis, de plus le spectateur est vite récompensé de sa fidélité contrairement à des séries type Mentalist ou Dexter, où on ne va pas assez vite au but. Ici en 12 épisodes réparties par 6 sur 2 saisons, on est vraiment accroché à l'histoire jusqu'au bout, on en démort pas, on aime chaque personnage même les plus "terribles". Pour le décor, nous sommes dans le Birmingham des années 20, avec des éléments qui rappelle la révolution industrielle. Les lieux sont magnifiques, vraiment des tableaux, à chaque point de vue on pourrait prendre une belle photo. C'est simple même le générique, Nick Cave - Red Right Hand, est un chef d'oeuvre. Vraiment vivement la saison 3, d'ici là je vous la recommande sans modération !

Auteur: Once Upon A Times - Note: 5 - Code commentaire: 45245898

Ladies and Gentlemen, may have your attention please ! L'hiver est froid, l'attente trop longue. Je compte désormais parmi mes séries de haute volée : Peaky Blinders. Une série non seulement bien jouée, qui repose sur une bande son rock Artic Monkeys et Nick Cave à l'honneur mais qui se dote d'un scénario béton. Oui un scénario, chose qui fait malheureusement défaut sur le paysage des séries actuelles. Une question: Pourquoi cette série n'est-elle pas plus connue ? Mystère, en attendant bon visionnage.

...

Auteur: Laura L. - Note: 5 - Code commentaire: 53725003

Que je suis contente de pouvoir enfin vous parler ici de Peaky Blinders ! Quand mon homme m'a proposé de commencer cette série, je savais qu'elle ne pouvait que me plaire. Et ce fut le cas des les premiers épisodes l'univers, la bande son et la mise en scène la combinaison est parfaitement huilée. J'ai du mal d'ailleurs à saisir qu'elle ne rencontre pas de succès populaire, un peu comme Boardwalk Empire à son époque elle mériterait bien plus ... Les saisons de Peaky Blinders sont composées seulement de 6 épisodes alors évidemment on rentre assez rapidement dans le vif du sujet tout en gardant cette atmosphère électrisante et pesante qui fait le charme de la série. Cette saison 3 ne dérogera pas à la règle chacun de ces épisodes est magique et apporte son lot de surprises et de scènes magistrales. Divers enjeux se mêlent et s'entremêlent à la perfection. Je reste pantoise par cette série qui finira par m'offrir contre toute attente un final épique. Je me suis retrouvée estomaquée par cette scène finale, l'ascenseur émotionnel est manié à merveille et certaines répliques ne peuvent que devenir cultes. Peaky Blinders, c'est aussi (et surtout?) des personnages charismatiques, le casting est vraiment bien constitué et chaque personnage à son évolution propre sans pour autant interférer avec leurs histoires personnelles sur la trame principale. Je suis tombée amoureuse du personnage de Thomas Shelby. Son côté sombre est ressorti cette saison nous permettant de découvrir d'autres facettes de celui ci et j'applaudis l'énorme prestation de Cillian Murphy. Paul Anderson nous offre aussi un Arthur toujours aussi génial, on ne peut qu'apprécier ce personnage qui est à la fois touchant et complètement déjanté. Mais mon évolution préférée est sûrement celle du dernier frère, John qui gagne énormément en prestance au fil des saisons et je pense que ce personnage est bien parti pour devenir encore plus central. En bref cette série est mon dernier AMOUR et cette saison 3 le confirme. Elle m'a tout bonnement chamboulée et je ne pense pas pouvoir citer une série qui la dépasse actuellement je suis même venu à me demander si elle ne finira pas par détrôner Breaking Bad dans mon cœur de séries addict ... Il faut absolument venir rencontrer la famille Shelby ... Il serait trop dommage de passer à côté d'une série de si bonne qualité.

Fichier traité: serie001_p2.html

Auteur: romano31 - Note: 5 - Code commentaire: 55940043

Peaky Blinders continue sur sa superbe lancée avec une saison 3 toute aussi réussie que les précédentes. Le clan Shelby est cette fois-ci confronté aux russes et cela ne va pas être de tout repos. Toujours aussi bien écrite, mise en scène et interprétée, la saison 3 de Peaky Blinders tient toutes ses promesses. Le clan Shelby est vraiment malmené durant cette saison mais ce n'est rien comparé à ce qui les attend dans la prochaine saison qui s'annonce déjà explosive. Bref, on a hâte de découvrir ça et c'est pour l'instant un sans faute pour Peaky Blinders qui enchaîne les saisons et les épisodes avec une redoutable insolence.

...

Fichier traité: serie002_p1.html

Code série: 21079 - Titre: Liar : la nuit du mensonge

Auteur: elriad - Note: 4.5 - Code commentaire: 57136432

parler des séries britanniques revient à dire quasiment la même chose: elles sont passionnantes. Pour la plupart, elles contiennent entre 6 et 8 épisodes, ce qui contraint le scénario à être dense et riche (à l'inverse des USA qui délitent leurs séries sur 6 saisons de 20 épisodes), elles parviennent à faire la part belle à la psychologie de leurs personnages sans pour autant négliger le suspense et l'évolution dramaturgique. "Liar" n'échappe pas à la règle et offre un suspense psychologique dense et inquiétant, porté par la petite bonne de "Downtown Abbey" autre série géniale anglaise, qui porte ce personnage de femme blessée avec intelligence et subtilité. Des séries comme on aimerait en voir plus souvent...

Activité 12 : Parser v3 (version finale des exercices: Identifiants de séries et de commentaires locaux)

On va enfin refaire exactement ce qu'on faisait à l'étape précédente (activité 11, Parser v2) en y apportant quelques subtiles différences.

- On va faire en sorte que dans le fichier XML corpus produit, les codes des séries ne soient pas laissés avec les codes d'origine provenant des fichiers sources mais soient renumérotés à partir de 1. Chaque nouvelle série rencontrée prendra comme code la valeur entière suivante, en commençant à 1 pour la première rencontrée. On appellera cette numérotation une numérotation locale.
- On voudra aussi que les codes commentaires soient numérotés de façon plus simple. On numérottera chaque commentaire avec le numéro de la série, suivi du souligné, suivi du numéro d'un numéro de commentaire local. Le numéro de commentaire local sera de 1 pour le premier commentaire sur la série concernée, et augmentera de 1 en 1 en prenant les valeurs entières.

Résumé :

Les codes de séries seront de la forme : xx (entier de 1 en 1 commençant à 1)

Les codes commentaires seront donc de la forme : xx_yy (avec yy entier de 1 en 1 commençant à 1 et xx le code de série)

```

1 <?xml version="1.0" encoding="utf-8" ?>
2 <CORPUS>
3 <SERIE_COMMENTAIRES>
4 <SERIE id="1">Peaky Blinders</SERIE>
5 <COMMENTAIRES>
6 <COMMENTAIRE id="1_1" auth="Silly Man" eval="5">
7   Bonjour ! Pour ma première critique, je me lance avec ce qui pour moi est un ch
8 </COMMENTAIRE>
9 <COMMENTAIRE id="1_2" auth="Once Upon A Times" eval="5">
10  Ladies and Gentlemen, may have your attention please ! L'hiver est froid, l'att
11 </COMMENTAIRE>
12 <COMMENTAIRE id="1_3" auth="Frank S." eval="5">
13  Tres bonne série que j'ai découvert par hasard je dois dire. C'est dans la même
14 </COMMENTAIRE>
...
90 <COMMENTAIRE id="1_29" auth="romano31" eval="5">
91  Après une première saison de qualité, les Peaky Blinders reviennent dans une deu
92 </COMMENTAIRE>
93 <COMMENTAIRE id="1_30" auth="ploux" eval="5">
94  Une véritable Perle.Le Créateur Steven Knight nous livre une Fascinante Fresque
95 </COMMENTAIRE>
96 </COMMENTAIRES>
97 </SERIE_COMMENTAIRES>
98 <SERIE_COMMENTAIRES>
99 <SERIE id="2">Liar : la nuit du mensonge</SERIE>
100 <COMMENTAIRES>
101 <COMMENTAIRE id="2_1" auth="elriad" eval="4.5">
102  parler des séries britanniques revient à dire quasiment la même chose: elles sor
103 </COMMENTAIRE>
104 <COMMENTAIRE id="2_2" auth="Aure L." eval="4">
105  Autour du thème d'un viol, "Liar" traite intelligemment du mensonge et de la mar
106 </COMMENTAIRE>
107 <COMMENTAIRE id="2_3" auth="ManoCornuta" eval="3.5">
108  La série représente une gageure: tenter de concilier un semblant d'intrigue poli
109 </COMMENTAIRE>
...
157 <COMMENTAIRE id="3_9" auth="Raptor Bonaparte" eval="5">
158  # Note Saison 4 : 19 / 20 - La meilleure saison ! Les dés sont jetés, les règle
159 </COMMENTAIRE>
160 <COMMENTAIRE id="3_10" auth="cylon86" eval="5">
161  La série américaine atteint des sommets, elle nous l'avait déjà prouvé au niveau
162 </COMMENTAIRE>
163 </COMMENTAIRES>
164 </SERIE_COMMENTAIRES>
165 </CORPUS>
166

```

Notre travail est fini si on utilise des fichiers de 2017. Mais si on veut des fichiers plus récents d'AlloCine il faut passer à l'étape suivante car le site ayant changé la structure de son contenu on ne peut plus utiliser le Parser v3. Or si on veut des grandes quantités de données, si on n'a pas d'archives datant de 2017 on ne pourra pas construire un ensemble de données assez grand.

Activité 13 : Parser v4 (version finale d'utilisation réelle en 2020) – donné en fin de séance

La structure des fichiers AlloCine en décembre 2020 (date de l'étude de cas actuelle) est très différente de ce qu'elle était en 2017. Donc si le fond de ce qui a été fait auparavant est correct, il faut changer les endroits où on récupère les données dans les fichiers à parser.

Où trouver les données ?

Balise <meta>, attribut "property", valeur="og:title" → Titre de série

Balise <meta>, attribut "property", valeur="apple-itunes-app" → Code de série

Balise <div>, attribut "id", valeur="review_..." → Bloc contenant la critique et ses informations annexes

Balise <div>, attribut "class", valeur="content-txt review-card-content" → Texte du commentaire critique

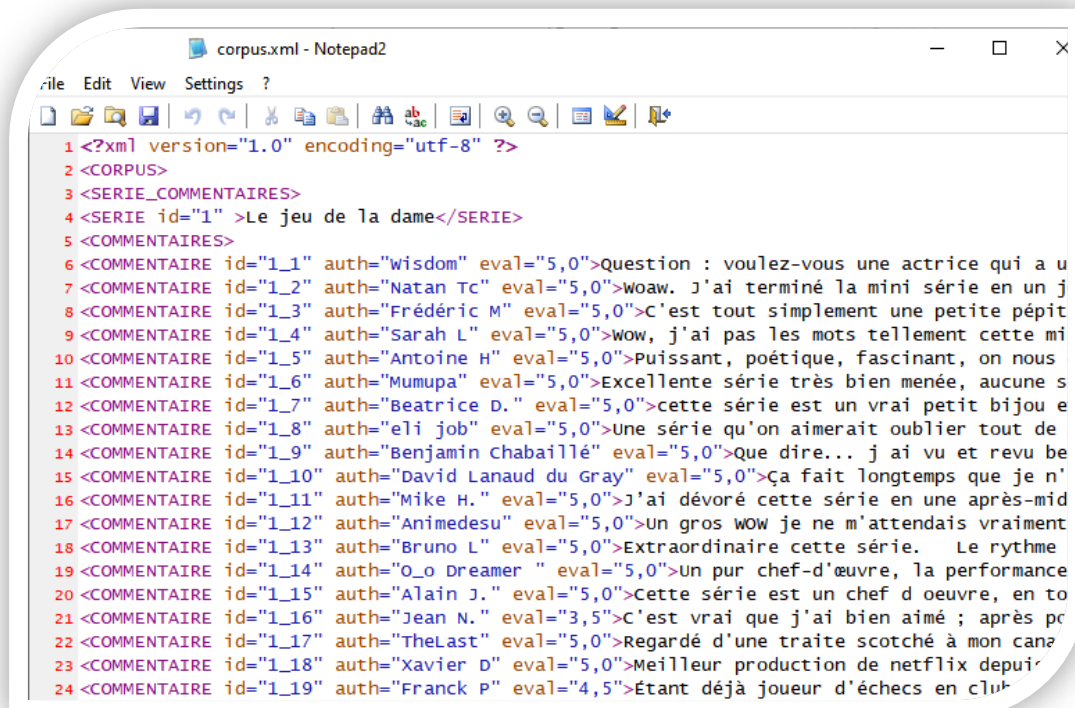
Balise <div>, attribut "class", valeur="meta-title", puis dedans, → Auteur du commentaire critique

Balise , attribut "class", valeur="stareval-note" → Evaluation laissée par l'auteur du commentaire critique

- On peut en profiter pour compter le nombre d'octets écrits dans le fichier XML afin de ne pas constituer des fichiers XML trop gros, disons de l'ordre de 5Mo et continuer dans un nouveau fichier corpus (on numérottera les corpus créés) pour la suite dès qu'on arrive aux 5Mo. On crée ainsi une série de fichiers corpus XML de 5Mo.
- On en profite aussi pour présenter un peu mieux les données en laissant sur une même ligne la balise de commentaire et son contenu, cela rend plus lisible le fichier corpus produit.

On pourra ensuite lancer la version finale de ce Parser v4 sur un ensemble de fichiers de critiques capturés en décembre 2020 et disponibles dans le dossier « [rep_captures_2020](#) » (contenant 28 fichiers pour 11Mo environ au total). Vous pourrez ainsi observer les fichiers XML produits. Si vous laissez la limite de constitution des fichiers XML à 5Mo, vous n'aurez qu'un seul fichier produit. Testez le rendu obtenu sur vos propres fichiers de captures provenant de l'aspirateur de site web de la veille.

Le rendu est le suivant sur un des fichiers produits :



```

1 <?xml version="1.0" encoding="utf-8" ?>
2 <CORPUS>
3 <SERIE_COMMENTAIRES>
4 <SERIE id="1" >Le jeu de la dame</SERIE>
5 <COMMENTAIRES>
6 <COMMENTAIRE id="1_1" auth="wisdom" eval="5,0">Question : voulez-vous une actrice qui a u
7 <COMMENTAIRE id="1_2" auth="Natan Tc" eval="5,0">Woaw. J'ai terminé la mini série en un j
8 <COMMENTAIRE id="1_3" auth="Frédéric M" eval="5,0">C'est tout simplement une petite pépit
9 <COMMENTAIRE id="1_4" auth="Sarah L" eval="5,0">Wow, j'ai pas les mots tellement cette mi
10 <COMMENTAIRE id="1_5" auth="Antoine H" eval="5,0">Puissant, poétique, fascinant, on nous
11 <COMMENTAIRE id="1_6" auth="Mumupa" eval="5,0">Excellente série très bien menée, aucune s
12 <COMMENTAIRE id="1_7" auth="Beatrice D." eval="5,0">cette série est un vrai petit bijou e
13 <COMMENTAIRE id="1_8" auth="eli job" eval="5,0">Une série qu'on aimerait oublier tout de
14 <COMMENTAIRE id="1_9" auth="Benjamin Chabailié" eval="5,0">Que dire... j ai vu et revu be
15 <COMMENTAIRE id="1_10" auth="David Lanaud du Gray" eval="5,0">Ça fait longtemps que je n'
16 <COMMENTAIRE id="1_11" auth="Mike H." eval="5,0">J'ai dévoré cette série en une après-mid
17 <COMMENTAIRE id="1_12" auth="Animesdesu" eval="5,0">Un gros wow je ne m'attendais vraiment
18 <COMMENTAIRE id="1_13" auth="Bruno L" eval="5,0">Extraordinaire cette série. Le rythme
19 <COMMENTAIRE id="1_14" auth="O_o Dreamer " eval="5,0">un pur chef-d'œuvre, la performance
20 <COMMENTAIRE id="1_15" auth="Alain J." eval="5,0">Cette série est un chef d oeuvre, en to
21 <COMMENTAIRE id="1_16" auth="Jean N." eval="3,5">C'est vrai que j'ai bien aimé ; après pr
22 <COMMENTAIRE id="1_17" auth="TheLast" eval="5,0">Regardé d'une traite scotché à mon cana
23 <COMMENTAIRE id="1_18" auth="Xavier D" eval="5,0">Meilleur production de netflix depuis
24 <COMMENTAIRE id="1_19" auth="Franck P" eval="4,5">Étant déjà joueur d'échecs en club

```

Pour effectuer un travail assez complet dans cette étude de cas, il faut beaucoup de données. Donc on pourra utiliser un aspirateur de site web personnalisé qui ne va aspirer de façon ciblée que les fichiers de critiques de séries et en quantité. Comme vous l'avez constaté l'aspirateur déjà tout fait n'est pas assez paramétrable pour qu'on puisse obtenir seulement les fichiers qu'on désire ni de façon exhaustive.

Le programme aspirateur PHP personnalisé (qui récupère donc exclusivement les données dont on a besoin) est fourni et s'appelle « [aspirateur.php](#) » et enregistre les fichiers dans le dossier « [rep_aspirateur](#) ».

Au final on aura pu avoir toutes les critiques de toutes les pages de critiques de toutes les séries sur toutes les pages listant les séries du site AlloCine grâce à cet aspirateur personnalisé. Il a été écrit en PHP en fonctionne sur le principe des fichiers parser expliqués jusque-là.

Ce que fait cet aspirateur personnalisé :

- Il va aspirer, à partir de la racine du site AlloCine pour les séries: <https://www.allocine.fr/series-tv/>
- Il va parcourir toutes les pages (981 pages au moment de rédaction de ce document) contenant les titres de séries cliquables sous forme de liens. Pour cela il accède aux pages avec une URL au format : <https://www.allocine.fr/series-tv/?page=xxx> (avec xxx : numéro de page de séries)
- Il va, pour chaque série de ces pages, aller voir s'il existe un fichier de critiques des spectateurs qui a le format : <https://www.allocine.fr/series/fichserie-xxxxx/critiques/> (avec xxxxx : code de série)
- Puis il va lire le nombre de pages de commentaires existant pour la critique des spectateurs de cette série (en lisant sur le bas de page au niveau des liens de parcours des pages) et il va parcourir chacune de ces pages de critique et l'enregistrer sur disque avec un nommage normalisé :
critiquesspectxxx_yyyyy-aaaaa_bbb.html
xxxx : n° de page de séries parcourue depuis la racine de AlloCine sur les séries.
yyyy : n° de série enregistré sur disque
aaaaa : code série AlloCine de la série enregistrée
bbb : n° de page de commentaires critiques de la série qui a été enregistrée

< PRÉCÉDENTE		1	2	3	SUIVANTE >	
4	5	6	7	8	9	10
20	30	40	50	60	70	80
90	100	200	300	400	500	
600	700	800	900	...	981	

Exemple : critiquesspect0002_00022-11042_001.html

Page des séries n° 0002 / n° de série enregistrée sur disque: 00022 / Code série : 11042 / Page : 001

C'est donc la 22^{ème} série dont les commentaires ont été enregistrés, de la page 1 des commentaires.

- Il a été conçu pour aspirer avec reprise : si il est interrompu (par le temps maximal d'exécution d'un script PHP, qu'il faut régler au maximum, soit 9999 secondes avec Wampserver = 2h et ¾ d'heures environ) il peut être relancé et va continuer son travail exactement là où il s'était arrêté ; rien n'est perdu ni à refaire.

On pourra donc relancer le Parser V4 sur le contenu de ce dernier dossier pour produire des fichiers de données parsées volumineux servant à la suite de l'étude de cas.

Fichiers de 5Mo produits :

Etude de cas > rep_sorties_xml

Nom	Modifié le	Type	Taille
corpus_1.xml	07/12/2020 20:36	Fichier XML	4 889 Ko
corpus_2.xml	07/12/2020 20:45	Fichier XML	4 884 Ko
...			

Une version du même fichier Parser v4 mais avec une fonctionnalité de sauvegarde du dernier n° de fichier XML produit, du dernier fichier source parsé et du dernier n° de série locale mémorisé a été faite. Elle permet de sauvegarder ces informations dans un fichier et de les reprendre au lancement si ces informations sont disponibles. Ceci permet de poursuivre le travail de parsing là où il s'était arrêté en cas d'interruption du programme PHP (qui a lieu quand la durée maximale d'exécution autorisée est atteinte, soit 9999s au max avec Wampserveur). Ainsi on peut traiter de grandes quantités de données en plusieurs fois

→ La version est le **Parser v5 (avec reprise)**