8 exercices Python sur les regex (web scraping & HTML)

Exercice 1 : Recherche d'un code d'erreur dans du HTML

Énoncé : Vous analysez le code HTML d’une page web pour vérifier si un code d’erreur HTTP y est présent. Par exemple, on souhaite détecter la mention « Erreur 404 » qui indiquerait une page non trouvée. Chaîne d'entrée : `html <html> <body> <h1>Erreur 404 - Page non trouvée</h1> <p>Retour à l'<a href="/">accueil</a></p> </body> </html>`
Consigne : Écrivez une expression régulière pour rechercher la sous-chaîne **"Erreur 404"** dans le contenu HTML ci-dessus, et utilisez-la (par exemple via `re.search`) afin de déterminer si le code *404* apparaît dans la page. (Indication : la méthode `search()` trouve une correspondance n'importe où dans la chaîne:contentReference[oaicite:0]{index=0}:contentReference[oaicite:1]{index=1}.)

Exercice 2 : Extraction des titres

d’une page HTML

Énoncé : On dispose d’un extrait de page HTML contenant plusieurs titres de niveau 2 (

). L’objectif est d’extraire toutes les balises

ainsi que leur contenu. Cela permet par exemple de récupérer les en-têtes de sections d’un article web. Chaîne d'entrée : `html <div> <h2>Introduction</h2> <p>Bienvenue sur notre site...</p> <h2>Contact</h2> <p>Envoyez un message via le formulaire.</p> </div>` Consigne : Écrivez une regex qui trouve toutes les occurrences de `<h2>...</h2>` (y compris le texte à l’intérieur). Veillez à ce que votre motif ne capture que le contenu de chaque titre sans empiéter sur les autres titres. *Exemple :* une quantification naïve `.*` est gloutonne et risque de tout englober d’un seul coup:contentReference[oaicite:2]{index=2}. Utilisez un quantificateur non-glouton (par ex. `.*?`) ou une classe de caractères appropriée pour limiter la correspondance au texte de chaque titre:contentReference[oaicite:3]{index=3}:contentReference[oaicite:4]{index=4}.

Exercice 3 : Extraction des attributs href des liens HTML

Énoncé : On veut extraire toutes les URL de lien présentes dans un extrait de code HTML. Plus précisément, pour chaque balise d'hyperlien <a ...>, il faut récupérer la valeur de son attribut href (l'adresse du lien). Chaîne d'entrée : `html <p>Retrouvez-nous sur les réseaux : <a href="https://twitter.com/compte">Twitter</a>, <a href="https://github.com/compte">GitHub</a> </p>` **Consigne :** Écrivez une expression régulière capturant la valeur de chaque attribut **href** dans ce code. Autrement dit, pour chaque `<a href="...">`, la regex doit extraire le texte entre guillemets après `href=`. Astuce : vous pouvez utiliser un **groupe capturant** pour isoler le contenu entre guillemets et le récupérer sans inclure les guillemets eux-mêmes.

Exercice 4 : Extraction des adresses email

Énoncé : Dans le cadre d'un scraping de texte, on souhaite extraire toutes les adresses email mentionnées dans un contenu textuel (par exemple, afin de collecter des contacts ou détecter des informations personnelles). Le texte peut contenir des adresses avec des formats variés (lettres, chiffres, points ou tirets dans le nom et le domaine). Chaîne d'entrée : `Pour toute question, écrivez à support@example.com ou bien à jean.dupont99@test.co.uk. Vous pouvez aussi visiter notre site web.` **Consigne :** Élaborez une regex qui **trouve toutes les adresses email** dans le texte. Utilisez des **classes de caractères** et des **quantificateurs** pour couvrir les cas typiques (suite de caractères alphanumériques, éventuellement avec `.` ou `-`, suivie d'un `@`, puis d'un nom de domaine):contentReference[oaicite:5]{index=5}. Par exemple, le motif pourrait accepter des caractères comme lettres, chiffres, point ou tiret avant et après le `@`. Utilisez `re.findall` pour obtenir la liste de toutes les adresses trouvées dans la chaîne:contentReference[oaicite:6]{index=6}:contentReference[oaicite:7]{index=7}.

Exercice 5 : Nettoyage des espaces superflus

Énoncé : On a extrait le texte brut d'une page web, mais celui-ci contient des espaces multiples et des retours à la ## ligne inutiles. Pour une meilleure présentation, on souhaite nettoyer ce texte en ne laissant qu'un seul espace entre les mots (et en éliminant les espaces/tabs en trop ou lignes vides).
Chaîne d'entrée : `Bienvenue sur notre site web. Veuillez vous connecter pour continuer.`
Consigne : À l'aide d'une regex de remplacement, nettoyez la chaîne pour qu'elle ne contienne plus qu'un seul espace consécutif et retirez les sauts de ligne superflus. Autrement dit, remplacez toute séquence de whitespace (espace, tabulation ou saut de ligne) de longueur ≥ 2 par un seul espace. Indication : le motif \s+ correspond à « une ou plusieurs occurrences d'un caractère blanc » (espace, tab, fin de ligne, etc.), ce qui permet de cibler les regroupements d'espaces geeksforgeeks.org . Utilisez re.sub pour effectuer la substitution sur l'ensemble du texte geeksforgeeks.org .

Exercice 6 : Anonymisation des emails dans un texte

Énoncé : Pour des raisons de confidentialité, on souhaite anonymiser les adresses email présentes dans un texte extrait du web. Plutôt que de les extraire, on veut les remplacer par un placeholder générique (par exemple, [EMAIL]). Ainsi, le texte pourra être diffusé sans révéler les emails réels. Chaîne d'entrée : `Utilisateur : Alice Email : alice.dupond@example.com Date d'inscription : 2023-09-15` **Consigne :** Écrivez une regex qui correspond à une adresse email dans le texte, et utilisez `re.sub` pour la **remplacer** par la chaîne `[EMAIL]`. Veillez à ce que *toutes* les adresses email présentes

soient anonymisées. (On peut réutiliser un motif similaire à celui de l’exercice 4 pour détecter les emails, puis les remplacer par le placeholder. **Hint :** la fonction `re.sub` appliquera le remplacement sur chaque occurrence trouvée.)

Exercice 7 : Utilisation de groupes capturants pour les dates

Énoncé : Un texte HTML contient une date au format **JJ/MM/AAAA** (jour sur 2 chiffres, mois sur 2 chiffres, année sur 4 chiffres). Par exemple, cela peut provenir d’un article de blog indiquant sa date de publication. On veut extraire séparément le jour, le mois et l’année.
Chaîne d'entrée : `<p>Article publié le <strong>23/07/2024</strong> sur notre site.</p>`
Consigne : Écrivez une expression régulière qui capture le **jour**, le **mois** et l’**année** dans la date. Utilisez des parenthèses pour définir trois **groupes capturants** correspondant respectivement au jour, au mois et à l’année. Par exemple, un motif possible est `(\d{2})/(\d{2})/(\d{4})` où `\d{2}` correspond à deux chiffres et `\d{4}` à quatre chiffres:contentReference[oaicite:10]{index=10}. Utilisez `re.search` ou `re.findall` puis les méthodes de l’objet Match (`group(1)`, `group(2)`, etc.) afin d’obtenir les trois éléments de date séparément dans votre code.

Exercice 8 : Extraction de texte entre guillemets (non-glouton)

Énoncé : Dans un texte issu d’une page web (par exemple une section de commentaires ou de script), on veut extraire toutes les portions de texte qui se trouvent **entre guillemets doubles**. Par exemple, extraire les phrases citées dans un paragraphe. Attention, la même ligne peut contenir plusieurs paires de guillemets, qu’il faut traiter séparément.
Chaîne d'entrée : `Il m'a dit : "Bonjour" puis il est parti en criant "Au revoir !".`
Consigne : Écrivez une regex qui trouve chaque segment de texte situé entre " et " dans la chaîne. Veillez à ce que le motif ne soit pas glouton, autrement il pourrait englober « Bonjour" puis il est parti en criant "Au revoir » en un seul match au lieu de deux. L’utilisation de *? (quantificateur non-glouton) permet de faire cesser la correspondance dès le premier guillemet fermant rencontré