*Partie 1 Saisir des données fengting ZHOU*

Introduction

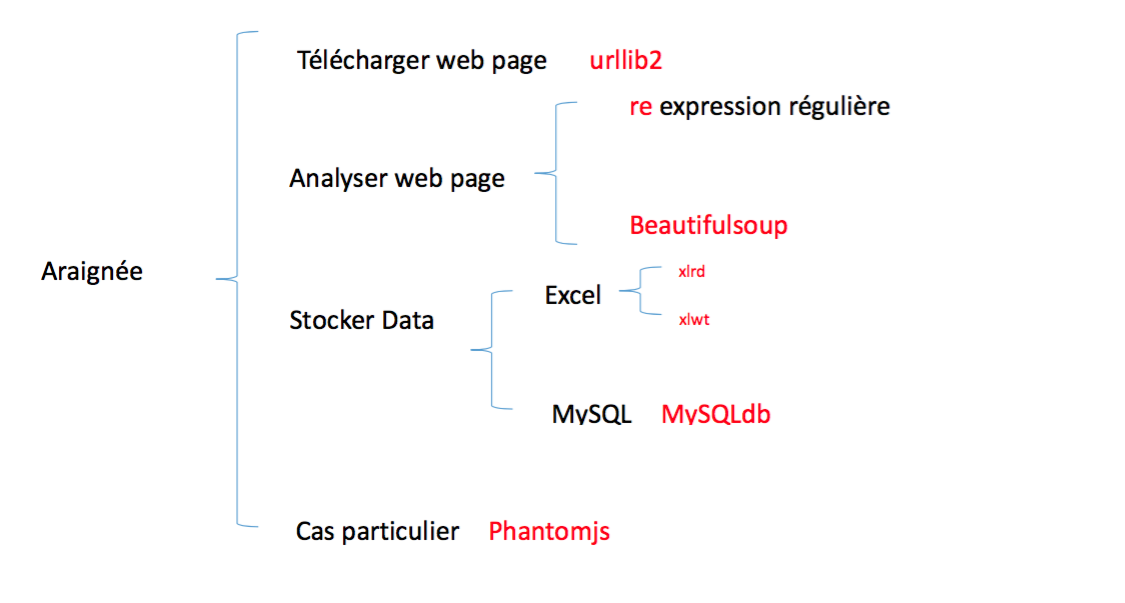
Chaque année 20 meilleurs équipes de football sont sélectionné par leur classements comme une composition « ligue 1 » en France. Pendant une saison de France ligue 1, chaque équipe va jouer deux match avec d’autre équipes, l’un comme « L’équipe home » et l’autre comme « L’équipe away ». Cela fait 380 matchs chaque année qui sont divisées par 38 « Journées », du coup chaque « Journée de match » on a 10 résultats de matchs qui sont joués par des équipes composés par différents ordres.

Puisque chaque match a son propre situation, on a besoin de prendre des paramètres particuliers en compte en plus d’un résultat simple, ce sont des «critères» qu’on utilise pour commencer à analyser des paris sportifs. Par exemple le calendrier (les équipes privilégient les matchs à domicile (surtout les petites équipes) car elles savent que si elles gagnent tous les matchs à domicile elles se maintiendront) ou la météo (Cela va jouer sur l’état du terrain et donc sur le jeu des équipes), etc.

Pour obtenir tous les données ci-dessus, on peut acheter le droit d’utilisation d’une base des données construit par des grandes entreprises statistiques, mais des services comme ça sont plutôt pour une autre entreprise et on n’a pas besoin de une base des données complète pour notre projet. Du coup on choisit de récupérer tous les données importantes en ligne par la programmation écrit par une loi d’une « araignée en ligne ».

Le logique d’une araignée en ligne

Normalement on construit une araignée d’après le principe décrit par ce tableau ci-dessous :

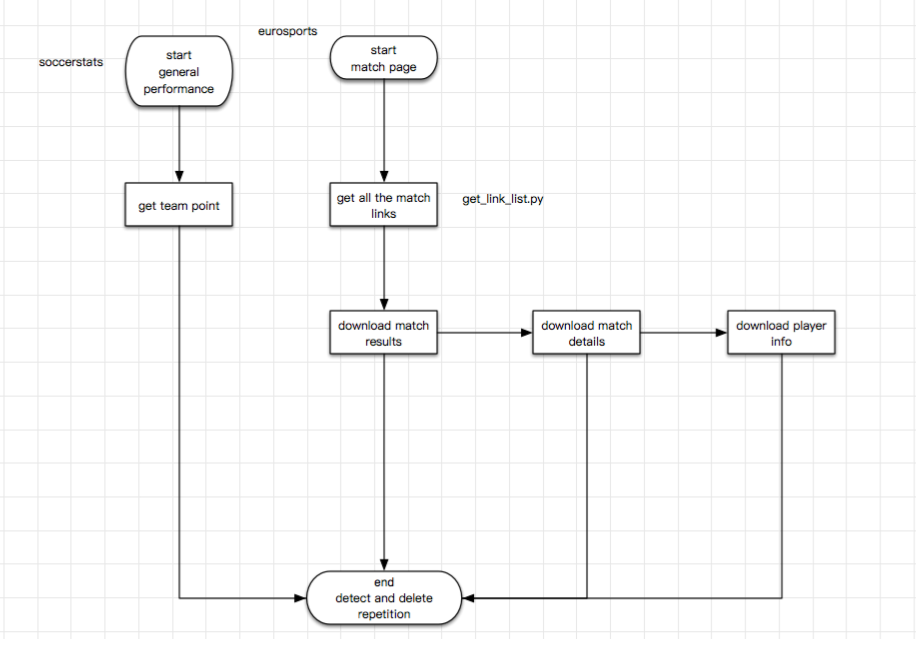


Puisque le langage Python est assez robuste pour les fonctionnements qu’on veut réaliser, je prends la structure de python et ses framworks comme un exemple. On utilise urllib2(la vielle version pour python2 mais plus fiable) tout d’abord pour faire lier le site web et le télécharger. Ensuite, avant de chercher des données qu’on a besoin, on doit analyser cette page de programme pour le reconstruire comme un « objet ». On peut aussi abandonner cette étape en utilisant la façon la plus fondamentale qui est « l’expression régulière », dans ce cas on copie les codes avant et après les données dans une expression pour dire l’ordinateur quelles sont des données qu’on a besoin. C’est n’est pas du tout efficace et on doit changer notre expression chaque fois on change la cible. Grace à le framwork Beautifulsoup, on peut utiliser ses APIs pour distinguer « le ticket » dans le code de front-end d’une page de site. Après on doit stocker les données dans n’importe quelle base dans notre ordinateur, plus souvent un Access dans l’environnement de windows ou Sql dans n’importe quel environnement.

Il existe une situation particulière ou les ingénieurs cache des données dans javascript pour des raisons de sécurité. Dan ce cas la python ne fonctionne plus et on doit écrire la programme en javascript et l’exploiter par un sans-tête navigateur « Phantomjs », dans lequel les codes imitent un navigateur et obtiennent les données comme une personne.

Diagramme de flux

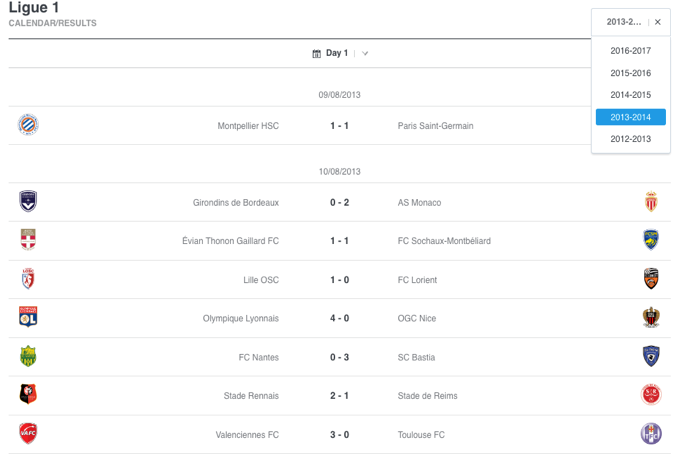
On trouve deux sites qui conviennent nos demandes. L’un s’appelle Eurosport et l’autre s’appelle SoccerStat. Dans le premier on saisit trois tableaux : les résultats annuels de 2013 à 2016 (Table\_1), les performances particulières (Table\_2) et les Compositions d’équipes (Table\_3), de plus on a trouvé les informations complémentaires sur ce site (La météo, le stade….. ). Dans le dernier on a besoin de 2 autres tableaux: les performances générales (Table\_4) et les performances particulières complémentaires (Table\_5). La diagramme de flux affiché en-dessous décrire l’ordre et le principe du araignée :



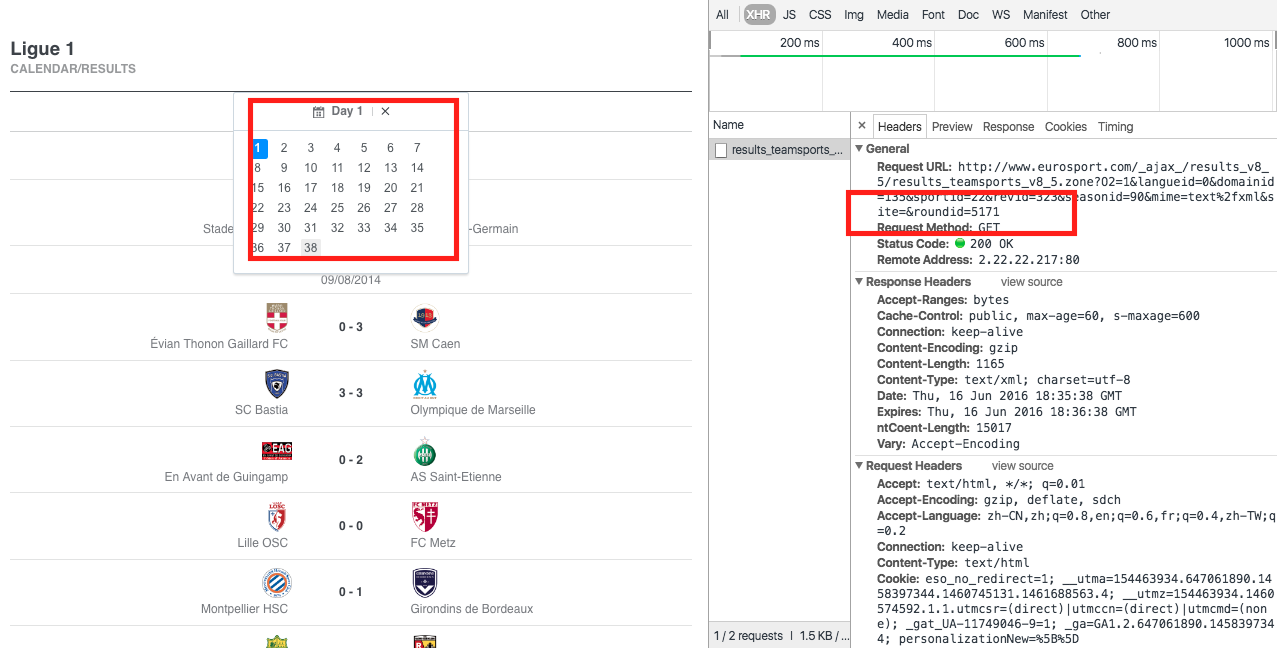
Réalisation en ligne

Un processus classique

D’abord je montre un processus typique pour créer une araignée. Le tableau 1 dans le site Eurosport conforme au tous standards comme un webpage, il affiche les résultats et l’ordre d’une saison de match.



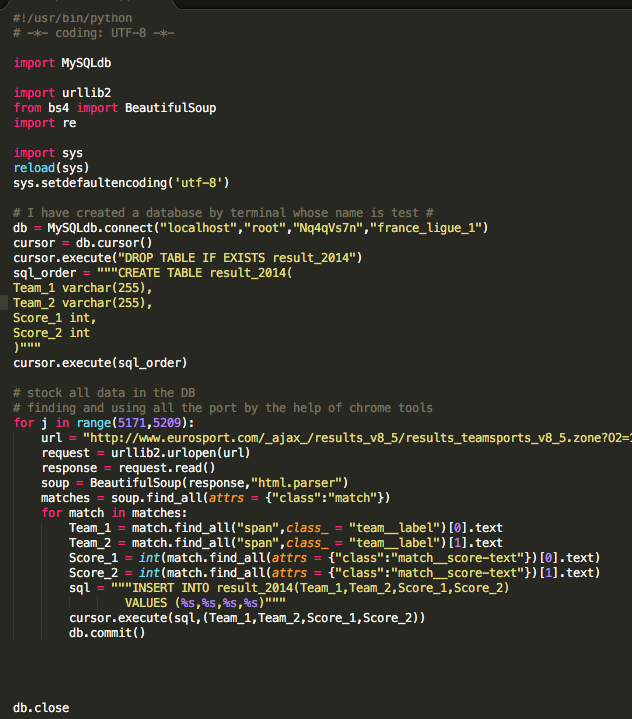
Pour faire le programme rendre visite à tous les journées de match automatiquement, on doit analyser ce page dynamique et collecte les adresses réelles dans lesquelles tous les données relatives sont stockées, parce que si on voit l’adresse affiché sur notre navigateur directement, on ne trouve pas de différence quand on changer la journée dans le formulaire, mais en fait il existe une liste des adresses réelles qui sont bien rangées par l’ordre. Donc on creuse cette liste en utilisant l’outil de chrome.

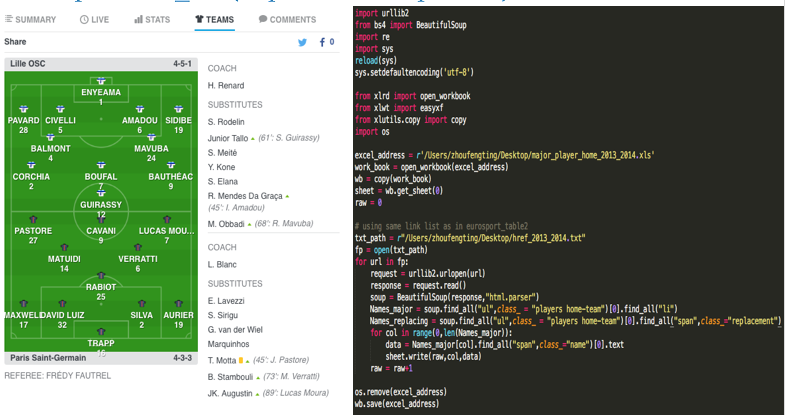


On peut voir que chaque année l’adresse est mit en ordre et le seul différent est le nombre dans la fin.

Après avoir cette liste, on fait un circulation dans le programme et grâce au beautifulsoup et les différent ticket des données, on peut trouvé ce que l’on veut sans prenant beaucoup d’effort.





Dans le code on peut voir que l’ordinateur distingue chaque données en cherchant son propre ticket comme « team\_lable », « score », etc. Ensuite on le stock dans MySQL. Tous les programmations sont écrites visuellement. On récupère le troisième tableau sur l’EuroSport identiquement, mais dans ce cas on le stock dans Excel. 

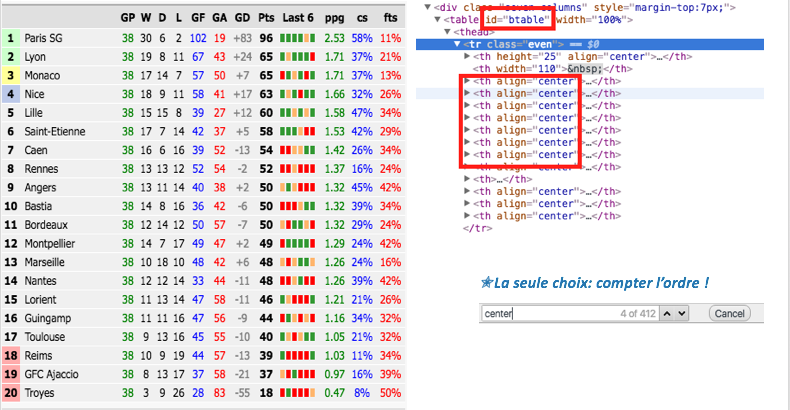
Informations complémentaire :



Avec des codes anarchiques

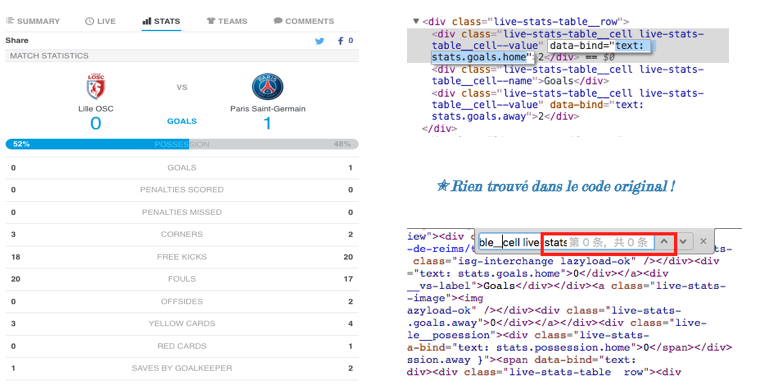
Ce processus agréable devient complexe si le web n’écrit pas trop bien. C’est le cas du website SoccerStat. Comme on peut voir en-dessous, il y a presque aucun ticket particulier pour chaque tableau et chaque colone des données (Chaque tableau s’appelle « btable » et il y a aucun ticket pour des paramètre de match !) L’ordinateur ne peut pas réaliser les fonctionnements s’il ne peut pas dire des différences entre le tableau que l’on a besoin et les autres tableaux enregistrés dans Soccerstat.

Il y a deux résolutions pour ce cas là, soit on abandonne ce site et chercher un autre(mais les données eux-même sont satisfaisantes du coup c’est n’est pas nécessaire), soit on compte l’ordre des tableau dans le code original jusqu’à trouver le bon tableau(il peut être dure) . Même principe pour chaque colone.

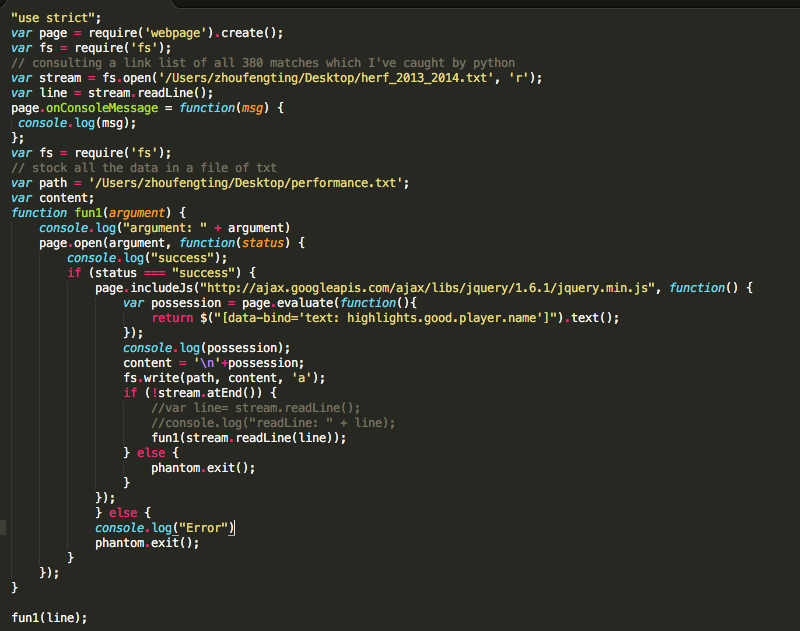


Cas particulier : render object

On rencontre le cas particulier dans le deuxième tableau de Eurosport, je trouve que même j’ai bien trouvé les tickets des différents paramètres pour chaque match, l’ordinateur n’ai rien trouvé dans la page comme les tickets n’existe pas. Donc je les vérifie dans le code original, et je trouve qu’il est écrit par une façon totalement différent, c’est à dire les données sont cachées dans javascript et il n’écrit pas comme ce qu’il affiche dans le page.



La seule résolution est le phantomjs que j’ai cité dans la deuxième partie du rapport (ou bien changer de nouveau un site mais je ne peux pas trouver un site pratique pour les paramètres particuliers). Du coup je construis un sans-tête navigateur en utilisant phantomjs et javascript et l’exploiter dans le terminal.



En fait c’est la mauvaise situation qu’on peut rencontre pour construire une araignée en cause de deux raisons : Premièrement, l’araignée construit par javascript n‘est pas chronoséquentiel ! On peut consulter le demo officiel pour écrire une circulation de python mais on ne peut pas faire ça dans javascript cars même on réussit de saisir une donnée, on ne peut pas le répéter en utilisant la répétition. Il va rapporter une erreur. On doit écrire une fonction et la met dans une récurrence pour assurer qu’il ne va pas finir jusqu’à la fin. Deuxièmement quand le terminal fait la exploitation, on ne peut pas faire d’autre chose puisque javascript n’est pas robuste, on est obligé d’attendre jusqu’à la fin d’un processus. Les deux raisons nous force de récupérer les données une par une. On peut voir dans le capture d’écran que il y a vers 10 paramètre pour chaque match, et on doit récupérer tous les match pendant 3 années, ça me prend une moi pour cette dernière partie !