Scrapy 1.7.2

Scrapy https://scrapy.org	**Framework de web crawling**	
	Système d'exploitation :	Multi-plateforme
	Développé par :	Scrapinghub, Ltd.
Licence BSD	Personne de contact :	Vandy.Berten@Smals.be

Fonctionnalités

Scrapy est un framework écrit en (et pour) Python permettant de faire du web-crawling (ou web-scraping), c'est-à-dire de l'extraction automatique de contenu à partir de pages web. Une grande partie du travail du web-crawling (ordonnancement des requêtes et parallélisation, chargement des pages, *parsing* du code HTML, rotation éventuelle de proxies…) est gérée automatiquement par le framework. Seul ce qui est spécifique au projet doit être géré par le développeur : choix des URL à charger, et, pour chaque (type de) page, informations à extraire. Pour des cas simples mais réalistes, le code (Python) à écrire peut se limiter à quelques lignes de code.

Scrapy se révèle très efficace pour le crawling de pages web dont le contenu à collecter est généré par le serveur web, et non pas le navigateur via Javascript. En d'autres termes, on pourra analyser le code HTML tel qu'il est reçu initialement par le navigateur, mais pas après modification par Javascript, comme le font certains sites populaires (Facebook, Google…). De notre expérience, cela ne pose cependant que rarement problème. Si c'était le cas, des solutions basées par exemple sur « Selenium » existent.

Scrapy shell, un environnement interactif en ligne de commande (s'utilisant comme iPython), permet une grande partie du travail « exploratoire » consistant en la sélection de combinaisons de tag HTML à rechercher pour extraire les informations désirées, ou les page suivantes à explorer.

Conclusions & Recommandations

Scrapy est un outil à la prise en main rapide, mais qui s'avère très efficace, tant en termes de temps de programmation qu'en terme de vitesse d'exécution. Il nécessite des connaissances de base en HTML/CSS.

Avant de démarrer un projet de webscraping, il conviendra cependant de se poser des questions par rapport à sa légalité et à son éthique : va-t-on collecter des informations personnelles ? Protégées par des droits d'auteur ? Risque-t-on de saturer le site visité ? Respecte-t-on ses conditions d'utilisation ? La finalité du projet et l'organisme pour lequel on le fait détermineront les problèmes bloquants.

Tests & Résultats

Le cœur d'un projet « Scrapy », quasiment la seule chose à coder, est constitué d'une fonction « parse » d'une classe dérivée de scrapy.Spider. Dans l'exemple ci-dessous, la page est constituée d'une séquence de citations, dans des blocs « ». Dans chacun de ces blocs, la citation elle-même est placée dans un bloc « », et le nom de l'auteur dans un bloc « ».

Cette fonction « parse » retournera principalement (via un « `yield` ») deux types d'objets:

- Soit des « données collectées », sous forme d'un dictionnaire Python (ici : `{'text' : …, 'author' : …}`). Chacun de ces objets est rajouté au résultat global du crawler, que l'on récupère typiquement sous forme d'un fichier « json ».
- Soit une ou plusieurs URLs, qui seront rajoutées au scheduler pour un traitement ultérieur, sauf si la page en question a déjà été traitée préalablement (ici : `yield response.follow(…)`)

```
def parse(self, response):
    for quote in response.css('div.quote'):
        yield {
            'text': quote.css('span.text::text').get(),
            'author': quote.xpath('span/small/text()').get(),
        }

    next_page = response.css('li.next a::attr("href")').get()
    if next_page is not None:
        yield response.follow(next_page, self.parse)
```

Exemple extrait de : https://docs.scrapy.org/

L'extraction d'information peut se faire au choix via deux standards : CSS ou XPath.

Dans l'exemple ci-dessus, on parcourt (`for quote in …`) l'ensemble des blocs « div » de classe « quote » (`response.css('div.quote')`). Dans chacun de ces blocs, on extraira le texte contenu dans un « span » de classe « text » (via le standard CSS : `quote.css('span.text::text')`), et le contenu d'un bloc « small » contenu dans un bloc « span » (via le standard XPath : `quote.xpath('span/small/text()')`).

Il est possible d'avoir plusieurs fonctions de parsing, par exemple pour différents types de page. On précisera à chaque « yield » renvoyant une URL (plus précisément un objet de type scrapy.Request ») la fonction à utiliser lors du parsing (ici `response.follow(…, self.parse)`).

De nombreux paramétrages sont possibles, selon que l'on désire être le plus discret possible, ou au contraire que l'on veuille atteindre les meilleures performances possibles : utilisation de proxies, nombre de requêtes en parallèle, délai entre les requêtes, respect ou non des fichiers « robots.txt »…

Conditions d'utilisation & Budget

Scrapy est gratuit et Open Source.