Chapitre 5
Analyser un historique de navigation web

1. Bibliothèques HTML et HTTP haut niveau

1.1 La bibliothèque requests

1.1.1 Présentation

Le package `requests` est une bibliothèque Python, distribuée sous licence Apache 2, qui permet d'effectuer des requêtes HTTP de manière confortable. L'unique version de HTTP supportée pour le moment est HTTP/1.1, qui est la plus largement utilisée. Il permet aussi d'utiliser SSL avec HTTPS, la version chiffrée de HTTP, de manière transparente.

HTTP (*Hypertext Transfert Protocol* : protocole de transfert d'hypertexte) est l'un des protocoles les plus utilisés d'Internet. C'est un protocole textuel (par opposition à binaire) qui permet l'envoi et la récupération de ressources depuis un serveur distant. Les échanges se font à partir de requêtes du client vers le serveur. Les requêtes sont composées de différents éléments. Au minimum une méthode, une URL et une version de HTTP. D'autres données peuvent être envoyées lors d'une requête, par exemple le nom de domaine ayant servi à contacter le serveur (permettant d'identifier un « site » par un nom de domaine et non par une adresse IP uniquement). De nombreuses informations supplémentaires sont regroupées dans ce qui constitue l'en-tête (*header*) de la requête.

Notamment les *cookies*, des données stockées côté client et mises à jour par les réponses du serveur, le champ `referer` contenant l'URL qui a provoqué la requête, le champ `user-agent` dans lequel le programme client décline son identité (nom du programme + numéro de version), l'encodage de caractère, etc.

Remarque

Les standards du web sont supervisés et publiés par le W3C (World Wide Web Consortium). Son rôle est d'instaurer des normes afin de garantir une compatibilité des technologies web. Sur leur site, il est possible de retrouver tous ces standards (voir section Hyperliens).

Le protocole définit un ensemble de méthodes associées, par convention, à un certain type d'action. Voici la liste des plus courantes :

- `GET` : qui demande une ressource identifiée par l'URL de la requête.
- `POST` : qui indique que la requête attend que la ressource référencée par l'URL traite les données présentes dans le corps de la requête (ce qui suit une succession de deux retours à la ligne après l'en-tête).
- `PUT` : qui demande le remplacement ou la création d'une ressource à l'URL indiquée, à partir du contenu du corps de la requête.
- `DELETE` : qui demande la suppression de la ressource présente à l'URL indiquée.
- `HEAD` : qui demande l'en-tête associé à la ressource, mais sans le contenu.

Remarque

Il existe d'autres méthodes comme `CONNECT`, `OPTIONS`, `TRACE`, `PATCH`. Ainsi que des extensions à HTTP (comme WebDav) qui ajoutent des commandes à celles définies par la RFC 7231 (voir section Hyperliens).

Les réponses du serveur sont constituées selon un format relativement similaire. L'en-tête comporte cependant plus fréquemment le champ `Content-Type`, destiné à renseigner le type MIME des données présentes dans le corps de la réponse.

Remarque

Le champ `Content-type` est aussi utilisé par les requêtes clients qui contiennent des données (typiquement `POST` ou `INSERT`).

Un des composants fondamentaux de la réponse d'un serveur est le « statut de la requête ». Celui-ci est représenté par un nombre compris entre 100 et 511. Les centaines regroupent des statuts par signification, les dizaines et les unités précisent la réponse :

– 1xx indique que la requête a été reçue et que le traitement continue.
– 2xx indique que la requête a été traitée avec succès.
– 3xx indique une redirection : la ressource se trouve à un autre emplacement.
– 4xx indique une erreur dans la requête du client.
– 5xx indique une erreur dans le traitement du serveur.

Exemples de codes HTTP

Le statut 200 indique que la requête a été traitée sans problèmes. Le statut 400 est renvoyé pour indiquer une erreur dans la requête du client, sans plus de précision. Si la ressource demandée n'existe pas, le célèbre statut 404 est envoyé. La valeur 503 signifie quant à elle que le service est indisponible.

1.1.2 Installation

Cette bibliothèque est assez diffusée pour être empaquetée par de nombreuses distributions GNU/Linux. Avec Debian, il faudra installer :

```
# apt-get install python3-requests
```

Elle est aussi référencée par Pypi et disponible pour l'installation via *pip* :

```
$ pip install requests
```

Une fois installée, il est possible d'importer le module `requests` :

```
>>> import requests
```

1.1.3 Envoyer une requête avec une commande spécifique

Le module `requests` propose un ensemble de fonctions permettant d'envoyer une requête HTTP/1.1 avec une commande spécifique. Ces fonctions sont nommées selon le nom de la méthode : `requests.get()`, `requests.post()`, `requests.head()`, `requests.put()`, `requests.delete()`, etc. Elles partagent un ensemble d'arguments communs (qui seront détaillés dans les sections suivantes) pour ajouter des éléments à la requête.

Elles partagent aussi le type de valeur de retour : des instances de la classe `Response` du module `requests.models`. Ces instances disposent de nombreux attributs et méthodes permettant de traiter la réponse du serveur. C'est le cas de l'attribut `header` qui contient un dictionnaire avec l'ensemble des champs présents dans l'en-tête de la réponse. L'attribut `text` permet, quant à lui, d'accéder aux données renvoyées par le serveur (dans une chaîne en fonction des indications de l'en-tête).

Remarque

La méthode `json()` permet de renvoyer l'objet Python associé à un contenu représenté au format JSON. Les arguments de cette méthode seront envoyés, à la fonction `loads()` du module `json` avec le contenu de la réponse.

Ces instances disposent aussi d'un attribut `request` qui contient une instance de la classe `Request` (du module `requests.models`) représentant la requête envoyée au serveur.

Remarque

Cet attribut permet notamment d'observer, après coup, l'en-tête de la requête via l'attribut `header`.

Téléchargement de la RFC 7230 à propos de HTTP

```
>>> rep = requests.get('https://tools.ietf.org/rfc/rfc7230.txt')
>>> rep.status_code
200
>>> print(rep.elapsed)
0:00:01.699461
>>> len(rep.text)
205947
>>> with open('/tmp/rfc7230.txt', 'w+') as rfcfd:
...     rfcfd.write(rep.text)
...
205947
```

Illustration d'échec de requêtes

Une requête invalide

```
>>> rep = requests.get('http://domaine.exemple/invalid')
>>> rep.status_code
400
>>> print(rep.text)
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
       <head>
               <meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
               <title>400 Bad Request</title>
       </head>
       <body>
               <div style="text-align:center;">
                       <h1>400 Bad Request !</h1>
               </div>
       </body>
</html>


>>> print(rep.elapsed)
0:00:00.051783
```

Une requête échouant sur un domaine impossible à résoudre en adresse IP.

```
>>> try:
...     rep = requests.get('http://invalid.dom/')
... except requests.exceptions.ConnectionError as expt:
...     print("Erreur : ", expt)
...
Erreur :  HTTPConnectionPool(host='invalid.dom', port=80): Max retries
exceeded with url: / (Caused by NewConnectionError('<requests.packages.
urllib3.connection.HTTPConnection object at 0x7f5ca74e5d30>: Failed to
establish a new connection: [Errno -2] Name or service not known',))
```

1.1.4 Passage de paramètres dans l'URL

Comme décrite dans la section dédiée à `urllib`, une URL peut être accompagnée d'un ensemble de paramètres. Ces paramètres sont fréquemment utilisés comme des arguments passés à une ressource dont le contenu dynamique est généré en fonction. L'exemple typique serait un annuaire disposant d'un système de pagination, avec des URL de la forme « http://hostname/annuaire?page=42 ». Le serveur se charge alors de renvoyer le contenu généré par la ressource (programme, script, etc.) qui aura été appelée avec un contexte contenant notamment l'URL et ses arguments.

Les fonctions décrites ci-dessus supportent nativement des URL contenant des arguments, mais disposent aussi d'un argument facultatif nommé `params` destiné à être un dictionnaire décrivant les paramètres de l'URL.

Deux requêtes identiques

```
requests.get('http://exemple.dom/a/b/c?arg=&arg2=val&arg3=v2')
requests.get('http://exemple.dom/a/b/c',
             params={'arg':'', 'arg2': 'val', 'arg3':'v2'})
```

1.1.5 Passage de paramètres pour une requête POST

D'une manière similaire à `params`, les fonctions utilitaires de requêtes disposent de l'argument optionnel `data`, cette fois destiné à représenter le contenu associé à la requête (passé dans le corps, après l'en-tête).

HTTP indiquant le type MIME du contenu dans la requête, des types de données très hétérogènes sont supportés. L'argument `data` peut donc, cette fois, être un dictionnaire ou une instance de `bytes` (une suite arbitraire d'octets). De plus, l'un des types définis par MIME, `multipart`, est conçu pour représenter un ensemble de fichiers.

Remarque

Le standard MIME (Multipurpose Internet Mail Extensions) était, comme son nom l'indique, conçu à l'origine pour les e-mails, technologie basée, elle aussi, sur des protocoles textuels.

Les fonctions de requêtes de `requests` disposent d'un argument `files` permettant d'envoyer une requête avec un contenu de ce type. Cet argument supporte différents types de données, mais attend typiquement un dictionnaire dont les clés sont les noms des fichiers et les valeurs associées sont des fichiers ouverts par la fonction `open()`.

Remarque

D'une manière identique aux arguments `data`, `params` ou `files`, les fonctions concernées disposent d'arguments similaires pour les champs de l'en-tête headers pour les cookies.