



Ressourcesinformatiques

Version numérique

OFFERTE !

www.editions-eni.fr

Python et l'analyse forensique

Récupérer et analyser les données
produites par les ordinateurs

Mehdi BENNIS
Yann WEBER

Fichiers complémentaires
à télécharger



Les éléments à télécharger sont disponibles à l'adresse suivante :
<http://www.editions-eni.fr>
Saisissez la référence ENI de l'ouvrage **RIPYTFO** dans la zone de recherche
et validez. Cliquez sur le titre du livre puis sur le bouton de téléchargement.

Avant-propos

Chapitre 1
Introduction : Python et l'informatique forensique

- 1. Pourquoi un livre sur le sujet ? 15
 - 1.1 La mutation de l'informatique forensique 15
 - 1.1.1 Des usages toujours plus nombreux et variés. 15
 - 1.1.2 Un cadre légal qui évolue à l'échelle planétaire. 16
 - 1.2 Une pratique qui se démocratise 17
 - 1.2.1 Des attentes en sécurité et en fiabilité
 toujours plus grandes 17
 - 1.2.2 De la nécessité de maîtriser ses outils 18
- 2. Présentation du langage 19
 - 2.1 Un langage adapté 19
 - 2.1.1 Python, un langage de script 19
 - 2.1.2 Une interface efficace avec C. 19
 - 2.1.3 Une communauté active 19
 - 2.2 Interprétation de Python 20
 - 2.2.1 CPython 20
 - 2.2.2 Autres implémentations 20
 - 2.2.3 URL 21
- 3. Le choix des logiciels. 21
 - 3.1 Le choix du système d'exploitation 21
 - 3.1.1 GNU/Linux. 21
 - 3.1.2 Le choix d'une distribution 22
 - 3.1.3 Les procédures d'installation dans ce livre 22

2 ————— Python et l'analyse forensique

Récupérer et analyser les données produites par les ordinateurs

Chapitre 2

Premiers pas en Python

1. Installation de CPython.....	23
1.1 Avec le gestionnaire de paquets de la distribution GNU/Linux.....	23
1.1.1 Sur GNU/Linux Debian 9	23
1.1.2 Autres gestionnaires de paquets	24
1.2 Autres méthodes.....	26
1.2.1 Depuis les sources.....	26
1.2.2 Avec les binaires fournis sur le site python.org	28
1.3 Exécuter du Python	28
1.3.1 Depuis l'interpréteur interactif	28
1.3.2 Depuis un fichier	29
1.3.3 Hyperliens.....	30
2. Éléments de syntaxe fondamentaux.....	30
2.1 Les commentaires.....	30
2.1.1 Commentaire monoligne.....	30
2.1.2 Les commentaires multilignes	31
2.2 Variables	31
2.2.1 Nommage	31
2.2.2 Affectation	31
2.3 Les fonctions.....	34
2.3.1 Appeler une fonction	34
2.3.2 Définir une fonction.....	35
2.3.3 Fonction lambda.....	36
2.4 Les objets et leurs instances.....	37
2.4.1 Rapide rappel sur la programmation orientée objet.....	37
2.4.2 Créer une instance	40
2.4.3 Accéder aux attributs, appeler une méthode	40
2.4.4 Définir une classe en Python	41
2.4.5 Les méthodes « magiques ».....	43
2.4.6 Le « modèle objet » de Python	46
2.4.7 Les attributs et les méthodes « de classe ».....	46

2.4.8	Hyperliens	49
3.	Les types de données essentiels	49
3.1	Les nombres	49
3.1.1	Entiers et flottants	49
3.1.2	Opérations sur les nombres	53
3.1.3	Opérateurs de comparaisons	55
3.1.4	Opérations bits à bits	56
3.1.5	Nombres complexes	56
3.2	Autres types	57
3.2.1	None	57
3.2.2	Booléens	57
3.2.3	Opérateurs booléens	58
3.2.4	Opérateurs d'identité	61
3.3	Itérables	63
3.3.1	Introduction	63
3.3.2	Les tableaux : list	63
3.3.3	Le dépaquetage de séquences	68
3.3.4	Les listes immuables : tuple	69
3.3.5	Les tableaux associatifs : dict	71
3.3.6	Les ensembles : set	72
3.3.7	Les itérateurs	73
3.3.8	Les fonctions de génération	76
3.3.9	Regrouper des arguments de fonctions dans un itérable	77
3.4	Les chaînes de caractères	80
3.4.1	Présentation	80
3.4.2	Fonctions et méthodes des chaînes de caractères	81
3.4.3	La méthode « format »	84
3.4.4	Formater une chaîne avec la syntaxe « printf »	86
3.4.5	Les chaînes d'octets	87
3.4.6	Encode et decode	89
3.4.7	URL	90

4 — Python et l'analyse forensique

Récupérer et analyser les données produites par les ordinateurs

4. Les structures de contrôles.	91
4.1 Conditionnels et boucles	91
4.1.1 Les branchements conditionnels	91
4.1.2 Les expressions conditionnelles	93
4.1.3 La répétition « tant que » avec while	94
4.1.4 L'itération avec for	95
4.1.5 Les instructions de contrôle de l'itération.	98
4.1.6 Le else de boucle	99
4.2 La compréhension de liste	99
4.2.1 L'expression d'une collection	99
4.2.2 L'introduction des conditions	100
4.3 Gestion des exceptions.	101
4.3.1 Introduction	101
4.3.2 Les classes d'exceptions de Python	103
4.3.3 Try, Except, Raise.	104
4.3.4 else/finally.	106
5. Mécanismes d'import.	109
5.1 Définitions	109
5.1.1 Les modules de Python.	109
5.1.2 Les packages en Python	109
5.2 Syntaxe de l'import	110
5.2.1 Le mot-clé « import »	110
5.2.2 Les chemins d'import	114
5.2.3 Le mot-clé « from »	115
6. Environnement, dépendances et communauté	116
6.1 Déploiement/installation.	116
6.1.1 Présentation	116
6.1.2 Le script setup.py	116
6.1.3 Egg et Wheel.	117
6.1.4 Pip	118
6.1.5 Gestionnaire de paquet d'une distribution	119
6.1.6 Hyperliens.	119

6.2	Les environnements virtuels avec virtualenv	120
6.2.1	Présentation et mise en garde	120
6.2.2	Installation et utilisation	121
6.3	Qualité du code	122
6.3.1	Les conventions de codage PEP8	122
6.3.2	Utilitaires	123
6.3.3	Hyperliens	124

Chapitre 3 Bibliothèque standard

1.	Bibliothèques utilitaires	125
1.1	Fonctions utilitaires	125
1.1.1	Afficher avec print	125
1.1.2	Interroger le type d'une variable	127
1.1.3	Obtenir de l'aide avec help	128
1.2	Modules utilitaires de Python	129
1.2.1	Garder une trace de l'exécution des programmes avec logging	129
1.2.2	Analyser les arguments de la ligne de commande avec argparse	131
1.2.3	Parallélisation facile avec multiprocessing.Pool	135
1.2.4	Hyperliens	137
2.	Opérations de base sur un système de fichiers	137
2.1	Se déplacer et explorer l'arborescence des dossiers	137
2.1.1	Les chemins d'accès : les paths	137
2.1.2	Le dossier courant cwd et le déplacement dans l'arborescence	138
2.1.3	Les fonctions utilitaires sur les chemins de os.path	139
2.1.4	Exploration d'une arborescence avec le module os	142

6 ————— Python et l'analyse forensique

Récupérer et analyser les données produites par les ordinateurs

2.2	Opérations sur les fichiers	144
2.2.1	Ouverture et fermeture d'un fichier	144
2.2.2	stdin, stdout & stderr	145
2.2.3	Lecture et écriture dans un fichier	147
2.2.4	Exemple : réaliser une image disque	151
2.3	Utilitaires	157
2.3.1	Manipulation des fichiers avec le module os	157
2.3.2	Aller plus loin dans la manipulation des fichiers avec shutil	160
2.3.3	Gérer les fichiers temporaires avec tempfile	160
3.	Opérations de base sur le réseau	164
3.1	TCP/UDP	164
3.1.1	Python et les sockets	164
3.1.2	Serveur	167
3.1.3	Client	170
3.1.4	Recevoir et envoyer des données sur un socket	170
3.1.5	Un faux serveur chargen	173
3.2	Le module urllib	177
3.2.1	Introduction	177
3.2.2	Analyse d'URL avec urllib.parse	179
3.2.3	Forger une URL avec urllib.parse	181
3.2.4	Hyperliens	182

Chapitre 4

Premiers pas dans l'analyse d'un fichier

1.	Le fichier dans son ensemble	183
1.1	Trouver le type d'un fichier avec les nombres magiques	183
1.1.1	Présentation	183
1.1.2	Installation de python-magic	184
1.1.3	Exemple d'utilisation	185
1.1.4	Hyperliens	187

1.2	Les fonctions de hachage	187
1.2.1	Présentation du concept	187
1.2.2	Le module hashlib de la bibliothèque standard	188
1.2.3	Contrôler l'intégrité des fichiers	189
1.3	Génération de "diff" avec diffliib.	194
1.3.1	Présentation de diff	194
1.3.2	Le programme GNU diff	194
1.3.3	Générer le diff de deux fichiers avec Python.	196
1.3.4	Générer des diff de fichiers binaires.	199
2.	Utiliser Python pour lire des métadonnées	206
2.1	De l'importance des métadonnées	206
2.1.1	Définition	206
2.1.2	Quels intérêts pour les métadonnées ?	207
2.2	Lire les informations fournies par un système de fichiers ...	208
2.2.1	L'espace utilisé	208
2.2.2	Les métadonnées temporelles avec os.path.	209
2.2.3	Le système de fichiers vu par le noyau	212
2.2.4	Identifier les liens matériels à l'aide des inodes.	215
2.2.5	Hyperliens	218
2.3	Métadonnées de fichiers multimédias avec XMP	218
2.3.1	Le choix du format de métadonnée XMP	218
2.3.2	Installation de python-xmp-toolkit	219
2.3.3	Utilisation de la bibliothèque	220
2.3.4	Présentation rapide de XML	222
2.3.5	Un petit explorateur de métadonnées	223
2.3.6	Hyperliens	232

8 ————— Python et l'analyse forensique

Récupérer et analyser les données produites par les ordinateurs

Chapitre 5

Analyser un historique de navigation web

1. Bibliothèques HTML et HTTP haut niveau	233
1.1 La bibliothèque requests.	233
1.1.1 Présentation	233
1.1.2 Installation	235
1.1.3 Envoyer une requête avec une commande spécifique ..	236
1.1.4 Passage de paramètres dans l'URL.	237
1.1.5 Passage de paramètres pour une requête POST	238
1.1.6 Hyperliens.	239
1.2 La bibliothèque BeautifulSoup	239
1.2.1 Présentation	239
1.2.2 Installation	240
1.2.3 Charger du HTML	240
1.2.4 Extraire du contenu	241
1.2.5 Hyperliens.	242
2. Les bases de données SQLite3	242
2.1 SGBD et SQL	242
2.1.1 Les Systèmes de Gestion de Base de Données	242
2.1.2 Structured Query Language (Langage de requête structurée).	243
2.1.3 Un ORM pour les interfacer tous	244
2.2 SQLite3 un SGBD léger	244
2.2.1 Présentation	244
2.2.2 Les types de données SQLite3	245
2.2.3 Hyperlien	245
2.3 Le module SQLite3	246
2.3.1 Connexion à une base de données.	246
2.3.2 Exécuter une requête SQL	247
2.3.3 Les curseurs.	248
2.3.4 Hyperliens.	250

3.	Accéder à l'historique d'un navigateur	251
3.1	L'historique des navigateurs les plus utilisés	251
3.1.1	Introduction	251
3.1.2	Mozilla Firefox	251
3.1.3	Safari®	253
3.1.4	Navigateur Android®	254
3.1.5	Navigateur Chrome™/Chromium™	255
3.1.6	Hyperliens	255
3.2	Écriture d'un module d'accès	256
3.2.1	Organisation générale du code	256
3.2.2	Exécuter la requête selon le navigateur	261
3.2.3	Exemple d'utilisation	266
3.2.4	Récupérer un document HTML	266
3.2.5	Mise en place d'un cache	267
3.2.6	Accélérer l'exécution avec multiprocessing	270
3.2.7	Hyperliens	273
3.3	Utilisations	274
3.3.1	Lister les noms de domaines visités	274
3.3.2	Établir des statistiques sur les en-têtes des documents	275

Chapitre 6

Partitionnement automatique de données

1.	L'apprentissage automatique ou apprentissage statistique	277
1.1	Présentation	277
1.1.1	Définition	277
1.1.2	« Supervision » des algorithmes	279
1.2	Le partitionnement non supervisé	281
1.2.1	Présentation de la famille d'algorithmes	281
1.2.2	Principaux algorithmes	281

10 — Python et l'analyse forensique

Récupérer et analyser les données produites par les ordinateurs

2. Bibliothèques Python	282
2.1 Calcul matriciel optimisé avec NumPy	282
2.1.1 Présentation	282
2.1.2 Installation	283
2.1.3 Le type numpy.array	284
2.1.4 Hyperliens	288
2.2 L'apprentissage automatique avec Scikit-Learn	288
2.2.1 Présentation	288
2.2.2 Installation	288
2.2.3 Le partitionnement de données avec scikit-learn	289
2.2.4 Hyperliens	289
2.3 Afficher des données avec Matplotlib	289
2.3.1 Présentation	289
2.3.2 Installation	289
2.3.3 Afficher un ensemble de données dans un graphique	290
2.3.4 Hyperliens	292
3. Une première application : regrouper les fichiers par date de création	293
3.1 Représentation des données	293
3.1.1 Description de la problématique	293
3.1.2 Stockage et identification des temps et des chemins	294
3.1.3 Écrire un script pour créer des instances de FileTime	300
3.1.4 Normalisation des données	301
3.2 Création du modèle et construction des sous-groupes	305
3.2.1 Préambule	305
3.2.2 Création du modèle avec MiniBatchKmeans	305
3.2.3 Création du modèle avec BIRCH	306
3.2.4 Création du modèle avec DBSCAN	307
3.2.5 Création des groupes à partir des modèles	308

3.3	Écrire une interface textuelle	309
3.3.1	Présentation de l'interface	309
3.3.2	Les fonctions utilitaires d'une interface textuelle	309
3.3.3	Une représentation des modèles pour l'interface	312
3.3.4	Le module ui	316
3.3.5	Afficher les données dans un graphique	320
3.3.6	Le menu principal et la boucle de traitement	324
3.3.7	Utilisation	327

Chapitre 7

Extraire les sujets d'un ensemble de textes

1.	Le traitement automatique des langues naturelles.	329
1.1	Introduction	329
1.1.1	Présentation	329
1.1.2	La modélisation de sujets.	331
1.2	Les langues naturelles en Python avec NLTK.	332
1.2.1	Présentation	332
1.2.2	Installation	332
1.2.3	Hyperliens.	333
1.3	La modélisation de sujets avec gensim	333
1.3.1	Présentation	333
1.3.2	Installation	334
1.3.3	Hyperliens.	335
1.4	Les autres bibliothèques utilisées dans ce chapitre.	335
1.4.1	Les caractères accentués avec unidecode.	335
1.4.2	La progression avec tqdm.	336
1.4.3	Hyperliens.	336

12 ————— Python et l'analyse forensique

Récupérer et analyser les données produites par les ordinateurs

2. Modélisation de sujet avec LDA et LSI	337
2.1 Extraire une liste de mots d'un texte.	337
2.1.1 Analyse lexicale ou tokenization.	337
2.1.2 Suppression des caractères accentués et de la ponctuation	338
2.1.3 Suppression des mots vides	338
2.1.4 Une première version du module « corpus »	339
2.2 Sac de mots et TF-IDF	340
2.2.1 Introduction	340
2.2.2 Génération d'un dictionnaire de mots.	341
2.2.3 Création de "sacs de mots"	341
2.2.4 Le modèle TF-IDF de gensim	342
2.2.5 Hyperliens.	343
2.3 Les modèles gensim pour LDA et LSI	344
2.3.1 Génération du modèle	344
2.3.2 Regrouper les documents par sujets	346
2.3.3 Récupération des sujets	347
3. Exemples d'application.	349
3.1 Depuis l'historique d'un navigateur web.	349
3.1.1 Réutilisation de l'analyse de l'historique.	349
3.1.2 Adaptation de l'interface d'extraction	349
3.1.3 Écrire un script doté d'une interface	351
3.2 Depuis des fichiers textes.	355
3.2.1 Utilisation de LibreOffice comme convertisseur	355
3.2.2 TF-IDF depuis un système de fichier	356
3.2.3 Un script et une interface	358
3.2.4 Hyperliens.	360

Chapitre 8

Inspection des processus du noyau Linux

1. Introduction	361
1.1 Ptrace et le noyau Linux	361
1.1.1 Exécutables et processus	361
1.1.2 La mémoire vue par un processus	362
1.1.3 Présentation de ptrace	363
1.1.4 Hyperliens	365
2. Inspection d'un processus avec Python	366
2.1 La bibliothèque python-pttrace	366
2.1.1 Introduction	366
2.1.2 Installation de python-pttrace	367
2.1.3 Hyperliens	367
2.2 « Tracer » un processus	367
2.2.1 Démarrer un nouveau processus	367
2.2.2 Attacher un processus existant	372
2.2.3 Mise en pause et reprise de l'exécution	374
2.2.4 Récupérer l'état des registres	380
2.2.5 Reprise d'exécution conditionnelle	383
2.2.6 Hyperliens	387
2.3 La mémoire d'un processus	388
2.3.1 Généralités sur l'organisation de la mémoire pour un processus	388
2.3.2 Les maps mémoire	390
2.3.3 L'interface d'accès à la mémoire de python-pttrace	392
2.3.4 Hyperliens	396
3. Exemple d'utilisation	396
3.1 Extraire le texte	396
3.1.1 Objectif du programme	396
3.1.2 La fonction d'extraction des chaînes de caractères	397
3.1.3 Récupérer les maps à partir d'un PID	399
3.1.4 Initialisation et appel	400

14 _____ Python et l'analyse forensique

Récupérer et analyser les données produites par les ordinateurs

3.2	Tricher à un jeu vidéo.	402
3.2.1	Présentation de l'approche.	402
3.2.2	La classe MemState	403
3.2.3	Un script et une interface pour MemState.	411
3.2.4	Exemple d'utilisation	413
3.2.5	Hyperliens.	418
	Index	419



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.

234 — Python et l'analyse forensique

Récupérer et analyser les données produites par les ordinateurs

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.

238 — Python et l'analyse forensique

Récupérer et analyser les données produites par les ordinateurs

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.