ESIEE Paris – Département Santé Energie Environnement
SEV-5204E - « e-health data : web services and information systems»

Tutoriel Web Services REST

PARTIE 1

1. Configuration et vérification de l' environnement de travail.

2. Envoi de requêtes http vers un agent serveur d'un Web Service existant et identification des principaux mécanismes implémentés lors de l'accès au contenu des ressources distantes.

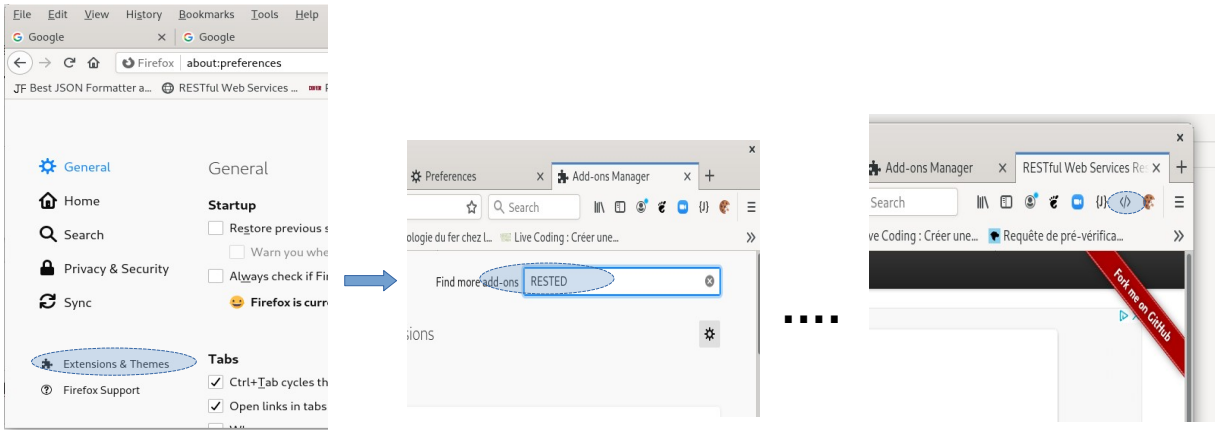
3. Ecriture en langage Python d'un agent client simple qui envoie des requêtes «GET» à un agent serveur existant, en utilisant la librairie Python «requests».

1 - Configuration et vérification de l' environnement de travail.

Pour suivre ce tutoriel, vous aurez besoin de :

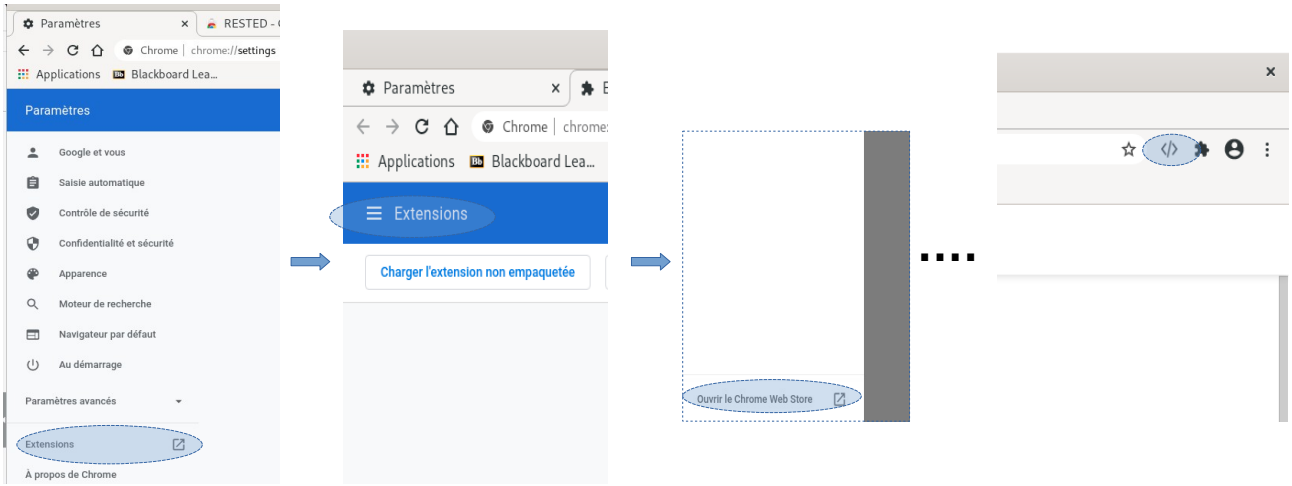
- **un PC sous Windows ou Linux ou un Mac sous OS X ;**
- **une connexion Internet** ;
- **un navigateur Web Firefox (ou Chrome) avec l' extension client REST «RESTED»** activée (obligatoire pour ce tutoriel / extension également disponible sur Opera mais pas sur Safari. Il est conseillé de privilégier l'utilisation de Firefox qui intègre nativement un moteur de rendu json qui permet de formater le jeu de données afin de le rendre plus lisible).

  Sur Firefox : Menu « *Edit/Preferences* » → click sur *Extensions & Themes* → *Find more-add-ons* → Firefox Browser ADD-ONS page → RESTED → Add to Firefox.
  Vous pouvez ensuite ouvrir un onglet client RESTED intégré en cliquant sur l' icône </> ajoutée dans la zone d' extensions de Firefox.



Sur Chrome :

Accédez à la page de configuration des paramètres de Chrome→ click sur *Extensions* → click sur *Extensions (top of the 2nd window)* →click sur *Open Chrome Web Store (bas de la 3ème fenêtre)* -> recherchez et ajoutez RESTED à Chrome. Vous pouvez ensuite ouvrir un onglet client RESTED en cliquant sur l'icône </> ajoutée dans la zone d' extensions de Chrome.



- **un IDE Python 3 (≥ python 3.5) (Spyder par exemple)**.
  NOTA : Il est fortement recommandé d' installer Anaconda3 Individual Edition car il comprend de nombreux packages utiles et l' IDE Spyder (voir https://docs.anaconda.com/)

- un gestionnaire de packages Python (pip recommandé), pour pouvoir installer des packages supplémentaires si nécessaire. L' installation d' un package supplémentaire Python peut se faire très simplement dans un terminal. Pip est automatiquement installé avec Anaconda et se trouve dans le répertoire :

  [anaconda_installation_directory]/bin

  La commande pour installer un package avec pip est :

  pip install [package_to_install]

  Si vous utilisez une distribution Python autonome, vous devriez pouvoir installer un package avec la commande :

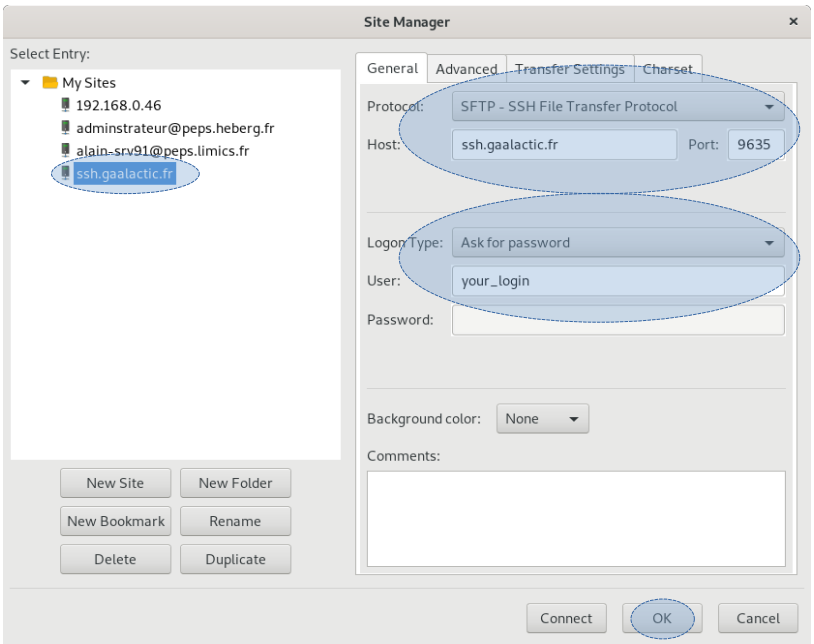
  python -m pip install [package_to_install]

- **une application de transfert de fichiers (WinScp recommandé sous windows ou filezilla sous Windows ou Mac OS X)** pour pouvoir téléverser tous les fichiers nécessaires (principalement des scripts python) sur le serveur http qui hébergera le service Web :

  https://filezilla-project.org/

  Une fois téléchargé et installé, lancez filezilla et exécutez la procédure de configuration de connexion du serveur comme indiqué ci-dessous :

  sélectionnez le menu « *File/Site Manager*» pour ouvrir la fenêtre du gestionnaire de site et remplir le formulaire. Vous devez choisir « *SFTP – SSH File Transfer Protocol*» pour pouvoir exécuter le transfert de fichiers via une connexion shell sécurisée qui est le seul mode autorisé par le serveur. Le nom de l' hôte doit être s*sh.gaalactic.fr,* le Port de connexion *9635,* le type de login *«Ask for password» et le nom de l' utilisateur doit correspondre à l'identifiant de connexion que vous avez reçue par e-mail. Donnez un nom à la connexion* (sss.gaalactic.fr for example) et cliquez sur OK pour sauvegarder la configuration. .



Procéder de façon similaire pour configurer **WinScp**.

2 - Envoi de requêtes à un agent serveur WS existant et identification des principaux mécanismes mis en œuvre lors de l'accès au contenu des ressources distantes.

Dans cette première partie du tutoriel, vous allez principalement utiliser l' extension RESTED du navigateur. Cette extension permet de simuler le comportement d'un agent client et de formuler et lancer des requêtes http de façon interactive. Vous allez découvrir ainsi comment on peut interagir, sans avoir à programmer, avec un agent serveur existant exposant une collection de ressources.

Dans l'exemple proposé, la collection regroupe des ressources représentant des messages destinés à différentes catégories d'individus. Chaque ressource contient deux éléments dont le premier (l'élément « dest ») représente la catégorie destinataire et le second (l'élémént « text ») contient le message correspondant. Le tableau ci-dessous est une des représentations possibles de la la collection des ressources :

dest	**text**
men	Hello men !
women	Hello women !
cats	Hello cats !
birds	Hello birds !

On imagine assez facilement que le contenu de ces ressources peut être hébergé dans une base de données, au sein d'une table comportant deux attributs « dest » et « text ». C'est ce qui sera fait dans le suite du tutoriel mais à ce stade, vous n'avez pas à vous préoccuper de savoir sous quelle forme le serveur stocke les données. Vous devez seulement vous intéresser aux modalités de représentation (json, html ou plain text) qui pourront être utilisées par le serveur pour renvoyer le contenu au client RESTED. Vous allez vérifier que le mode de représentation dépend de la directive « Accept » spécifiée par le client dans l'en-tête de la requête http.

★★ Travail à faire :

La première requête que vous allez exécuter adressera la racine de la hiérarchie des ressources exposées. Le jeu de données renvoyé à la suite de cette requête permet de connaître les caractéristiques de l' agent serveur. Vous pourrez ainsi savoir comment on doit interagir avec lui pour pouvoir récupérer les ressources exposées. Il est important de noter que tous les services Web REST devraient être capables d'envoyer ce type d'information pour permettre aux clients de savoir sous quelle forme et sous quelles conditions (accès libre ou restreint, en lecture seul ou pas, nécessitant une authentification ou pas, etc.) les ressources peuvent être consommées.

Dans l' exemple sur lequel vous travaillez, l' URI de la racine est :

https://www.gaalactic.fr/~lacombea/ws

Copiez l' URI dans une fenêtre de navigateur Web et exécutez la requête http. La réponse doit contenir un jeu de données de description du Web Service au format json qui ressemble à celui ci-dessos, qui a été correctement indenté et coloré afin de le rendre plus lisible.

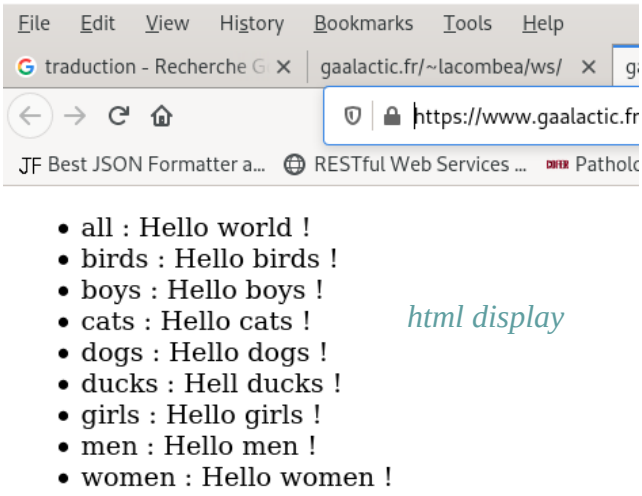

```
{
  "01-info":
      {
            "version": 1,
            "collection_name": "messages",
            "collection_URI": "https://www.gaalactic.fr/~lacombea/ws/messages",
            "title": "Messages Web Service",
            "host": "www.gaalactic.fr",
            "description": "Returns targeted Hello messages",
            "base_path": "/~lacombea/ws"
      },
  "02-methods":
      {
            "POST": "no",
            "DELETE": "no",
            "PUT": "no",
            "GET": "yes"
      },
  "03-headers":
      {
            "consume":
              {
               "Content type": ["application/x-www-form-urlencoded"],
               "Accept": ["application/json","text/html"]
              },
            "produce":
              {
               "Content type": ["application/json","text/html"]
              }
      }
}
```


Examinez de près le jeu de données. Vous pouvez noter qu'il contient toutes les informations utiles pour interagir avec la collection de ressources exposée par l'agent serveur :

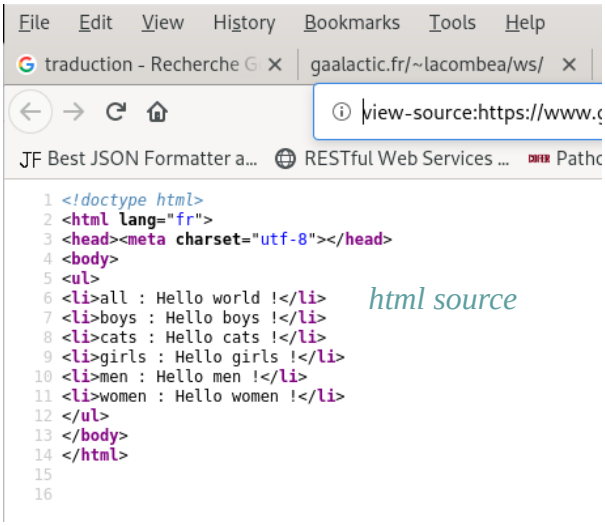
- ***01-info*** : informations générales sur l' agent serveur WS et sur la nature des ressources exposées ;
- ***02-methods*** : liste des méthodes de requête http autorisées / interdites - indique le type d' opérations qu' un client peut appliquer sur le contenu des ressources exposées (Create, Read, Update or Delete ?) ;
- ***03-headers*** : composé des deux éléments *consomme* et *produit* : le premier contient tous les en-têtes nécessaires au serveur pour pouvoir traiter correctement les requêtes http du client; le second donne la liste des en-têtes spécifiques renvoyés par l' agent serveur dans la réponse http.

★★ Travail à faire :

Recherchez dans l' ensemble de données l' URI permettant d'accéder à la collection des message. Copiez l'URI dans la barre d'adresse d'une fenêtre de navigateur Web standard et exécutez la requête http. Vous devriez recevoir un contenu qui ressemble à celui affiché en haut de la page 6.



html display

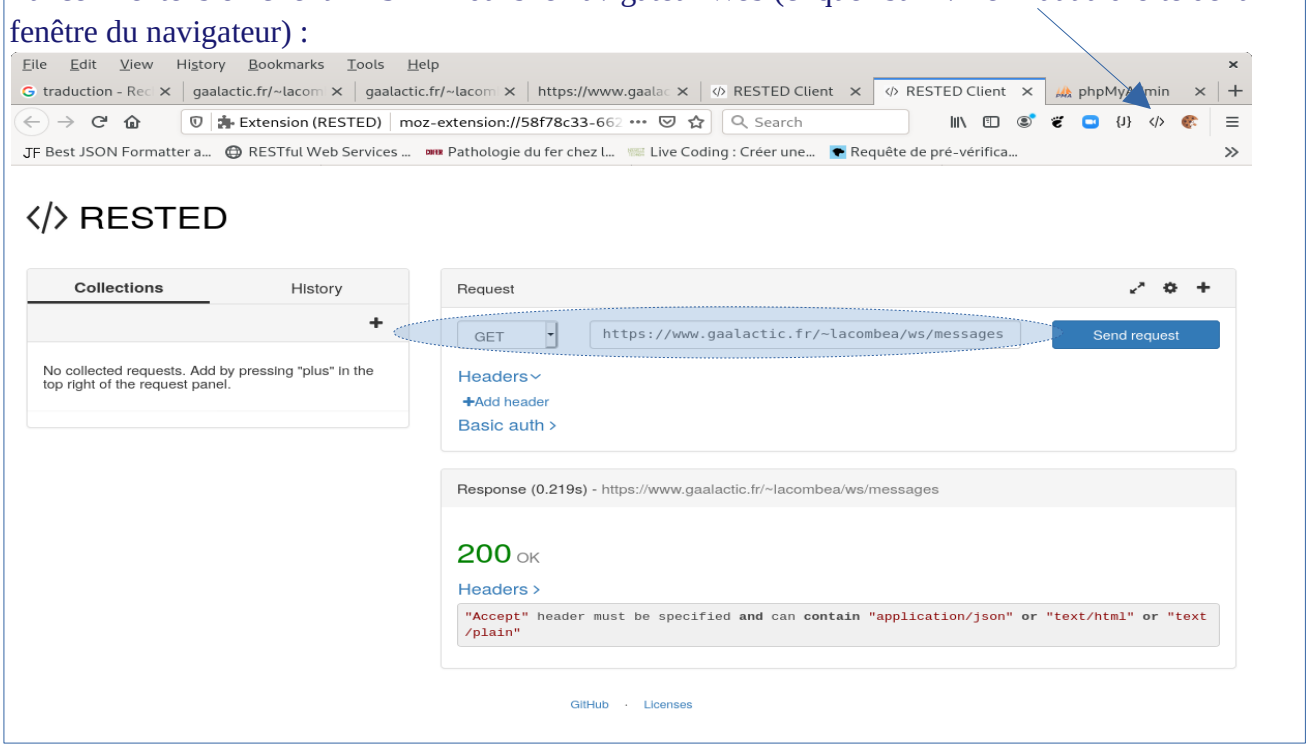


html source

Vous pouvez constater que lorsqu'une requête http est envoyée depuis une fenêtre standard du navigateur Web, la représentation des ressources renvoyées par l' agent serveur est formatée en html. Ce n'est pas un hasard et cela est dû au fait que le navigateur Web a implicitement envoyé une directive pour informer l' agent serveur du format de représentation qu'il est en mesure de consommer. Si vous pouviez visualiser le contenu de l' en-tête de la requête, vous constateriez qu'elle contient une directive *Accept* qui contient la spécification *text/html*.

★★ Travail à faire :

Lancez l'extension client RESTED dans le navigateur Web (cliquez sur </> en haut à droite de la fenêtre du navigateur) :



Remplissez le champ URI avec l' URI de la collection de messages et exécutez une requête GET (qui est équivalente à celle exécutée plus haut dans une fenêtre de navigateur Web standard).

Vérifiez et analysez le message d' erreur renvoyé par l' agent serveur. Comme vous pouvez le voir, un en-tête « Accept » doit être spécifié pour indiquer à l' agent serveur le type de représentation de

la ressource attendu par le client (*application/json* ou *text/html*), comme cela est spécifié dans l' élément *consume/Accept* du jeu de données descriptif (page 5).

Cliquez sur *Headers>* dans la fenêtre RESTED et ajoutez une directive d' en-tête dont le nom est *Accept* et la valeur est *text/html*. Lancez la requête, vérifiez la réponse et comparez son contenu au code source html reçu à la question 2.

Remarque: les valeurs de l' en-tête «Accept» doivent respecter la syntaxe de définition «MIME» contenue dans le jeu de données. Le standard MIME (Multipurpose Internet Mail Extensions) a été développé à l' origine pour permettre le transport de contenus hétérogènes (vidéo, photos, fichiers binaires,….) dans des courriers électroniques et a été étendu par la suite à toutes les applications web. Vous trouverez des informations détaillées sur ce standard à l' adresse:

https://developer.mozilla.org/fr/docs/Web/HTTP/Basics_of_HTTP/MIME_types/Common_types

2-4 Consommer une ressource individuelle.

Les requêtes http précédentes ont permis de récupérer toutes les ressources de la collection de messages exposée par le Web Service. Il est également possible d' envoyer une demande de récupération d'une ressource individuelle dont le contenu correspond au message destiné à une seule catégorie d'individus. Pour ce faire, il vous suffit de compléter l' URI précédent avec un paramètre qui correspond au nom de la catégorie souhaitée :

- *https://www.gaalactic.fr/~lacombea/ws/messages/boys*
- *https://www.gaalactic.fr/~lacombea/ws/messages/women*
- *https://www.gaalactic.fr/~lacombea/ws/messages/birds*
- ...

Vous pouvez également essayer d' envoyer une requête GET sur une ressource individuelle n'existe pas (par exemple *snakes* ou de spécifier une des méthodes http POST, PUT et DELETE. Dans ces deux cas, vous pouvez constater que l'agent serveur renvoie un message d'erreur composé d'un code et d'un texte qui documentent explicitement l'erreur qui s'est produite. Pour que cela soit possible, il a fallu prévoir, dans le script de l'agent serveur, une structure de gestion des différentes exceptions susceptibles de se présenter. Rappelez-vous que lorsqu'une exception n'a pas été anticipée, l' agent serveur peut avoir un comportement imprévu et renverra probablement un message d'erreur très difficile à interpréter par le client.

Rappelez vous que répertorier et prévoir la gestion des exceptions est un travail indispensable pour assurer la prédictibilité, la sécurité de fonctionnement et la facilité d'utilisation d'un Web Service.

En résumé.

Grâce à ces différentes opérations, vous avez appris comment accéder à la représentation d'un contenu exposé par l'agent serveur d'un WS sous la forme d'une collection de ressources. Vous avez vu que le comportement de l'agent serveur et les modalités suivant lesquelles il est possible d'interagir avec lui sont documentées par un jeu de données accessible à la racine du WS. Ce jeu de données contient notamment le ou les formats de représentation selon lesquels il peut retourner les contenus. Les navigateurs Web prennent au moins en charge le format *text/html* par défaut.

Un client logiciel Web développé à partir de zéro (un script Python par exemple) se comportera quant à lui comme le client RESTED et devra fournir explicitement toutes les informations attendues par l' agent serveur. Ces informations doivent être spécifiées par l'intermédiaire de directives insérées dans l'en-tête des requêtes http, comme la directive *Accept* par exemple.

Enfin, un programme d' agent serveur doit être capable de gérer les exceptions survenant en raison d' une mauvaise spécification d' en-têtes, d' un contenu d' URI de ressource incorrect ou de méthodes http (GET, POST, …) non autorisées. D'autres contrôles sont la plupart du temps nécessaires, comme le contrôle d'authentification ou des droits d'accès aux ressources par le client et seront abordés ultérieurement.

3 Ecriture en langage Python d'un agent client simple qui envoie des requêtes «GET» à un agent serveur existant, en utilisant le package Python «requests».

Il est temps pour vous d' écrire vos propres agents clients et serveurs en langage Python.

Le premier programme que vous devez écrire est assez simple. Il consiste en un agent client qui ne peut envoyer que des requêtes http GET permettant de consommer les ressources exposées par le serveur existant. Ce programme interrogera les ressources comme le fait l' extension RESTED, mais l' intérêt majeur est que les données renvoyées dans la représentation json pourront ensuite être utilisées et traitées par le programme client ou enregistrées pour un traitement ultérieur.

De la même manière qu' un navigateur Web intègre les fonctions logicielles de la couche réseau "application" pour gérer le protocole http, un agent client créé à partir de zéro doit s' appuyer sur des librairies spécifiques pour pouvoir requêter des ressources Web et récupérer les réponses renvoyées par les agents serveur. En Python, il existe plusieurs packages de librairies capables d' implémenter le protocole http, dont la librairie «requests». Cette librairie est assez facile à utiliser car elle contient un ensemble de classes et de fonctions dont le niveau d'abstraction permet une implémentation très intuitive.

Avant de commencer à écrire votre programme, vous devez vérifier si le package «requests» est installé (cela devrait être le cas si vous avez installé la distribution anaconda3). Pour savoir si «requests» est déjà installé, il suffit d' exécuter la commande ci-dessous dans un terminal:

> ***pip list*** ou ***> python -m pip list***

Cette commande affiche la liste et le N ° de version de tous les packages supplémentaires installés. Si «requests» est déjà présent, vous n'avez rien à faire de plus. Sinon, exécutez simplement:

> ***pip install requests***

Vous trouverez ci-dessous un exemple de programme Python lançant une requête GET qui demande le renvoi du contenu de la collection complète des ressources exposées par l'agent serveur existant.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Nov 20 18:10:25 2020
@author: lacombea
"""
import requests
import json

# URI d'accès à l'intégralité de la collection
uri = "https://www.gaalactic.fr/~lacombea/ws/messages"

# Dictionnaire contenant les données à insérer dans l'en-tête de la requête http
# Seul l'en-tête « Accept » est obligatoire. L'en-tête "User-agent" a
# été rajouté pour illustrer la façon dont on spécifie plusieurs directives d'en-tête
customHeaders = {"Accept": "application/json", "User-agent": "my_client_agent/v0.0.1"}

# Envoi de la requête http à destination de l'agent serveur
httpReturn = requests.get(uri, headers=customHeaders)

# Extraction du contenu du corps de la réponse (httpReturn.text)
# Dans le cas présent celui-ci correspond au contenu de la collection au format json
# puisque la valeur de l'en-tête « Accept » de la requête était "application/json"
representationContent = httpReturn.text
print("\nContenu du corps de la réponse http :")
print(representationContent)

# Désérialisation de la chaîne json et créer un objet qui contient
# les données sous une forme structurée
structuredRepresentationContent = json.loads(representationContent)

# L'objet ainsi obtenu correspond un dictionnaire
print("\nType de la structure contenant les données de la collection :")
print(type(structuredRepresentationContent))
print("\nContenu structuré de la collection :")
print(structuredRepresentationContent)
```

★★ Travail à faire :

- Copiez le script ci-dessus et enregistrez le dans votre espace de travail puis exécutez le et observez le résultat obtenu.

- Complétez le pour faire afficher le dictionnaire contenant les données du premier élément de la collection retournée.

- Modifiez le code pour permettre de récupérer la ressource contenant le message correspondant à une catégorie spécifique d'utilisateurs (men, women, cats,…).

- En vous aidant de la documentation accessible à l'adresse :

https://fr.python-requests.org/en/latest/user/quickstart.html#en-tetes-des-reponses ,

ajoutez le code nécessaire pour faire afficher le contenu de l'en-tête de la réponse http.

- En vous aidant de la documentation accessible à l'adresse :

https://fr.python-requests.org/en/latest/user/quickstart.html#codes-de-retour-des-reponses-status ,

- Ajoutez le code nécessaire pour faire afficher le code de retour correspondant au statut de la réponse http et recherchez la signification de ce code sur le site :

https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

- Modifiez la variable *url* pour simuler une erreur de syntaxe sur le nom de la collection (par exemple changez la valeur *https://www.gaalactic.fr/~lacombea/ws/messag*) de manière à envoyer une requête vers une ressource inexistante. Observez le message renvoyé par l'agent serveur et comparez le code de retour au code correspondant en vérifiant sa signification sur le site ci-dessus.

Modifiez le script pour traiter le code de retour juste après la réception de la réponse http, de manière à ce qu'il gère correctement le retour de l'agent serveur dans le cas où la ressource n'a pas pu être trouvée.

Note : afin d'apprendre à utiliser le package «requests», vous pouvez consulter la documentation disponible à l'adresse :

https://fr.python-requests.org/en/latest/user/quickstart.html

Une version approfondie de cette documentation est également disponible à l'adresse :

https://requests.readthedocs.io/en/master/

Vous devez également importer la librairie de modules json dans votre programme (package intégré) pour pouvoir désérialiser le contenu du corps de la réponse htpp lorsqu'une représentation json est renvoyée. La documentation sur le package json est disponible à l'adresse :

https://docs.python.org/fr/3/library/json.html

Recommandation *: pour débogger vos scripts, il peut être utile d'en interrompre l'exécution à plusieurs endroits successifs pour contrôler l'évolution du contenu des variables et d'identifier plus facilement la partie du code susceptible de provoquer une erreur d'exécution. Vous pouvez arrêter un script en cours de route en important la librairie* ***sys*** *et en insérant la fonction* ***sys.exit()*** *à l'endroit où le script doit s'interrompre.*