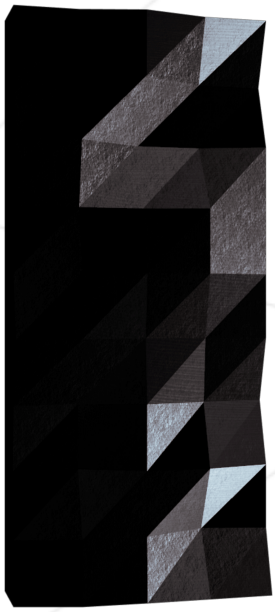

D03 - Formation Python-Django

English version coming soon !

Résumé: Aujourd'hui, nous allons manipuler quelques librairies pratique en **`Python`**.

Table des matières

Chapitre I

Préambule

Geohashing From Wikipedia, the free encyclopedia

Geohashing is an outdoor recreational activity inspired by the xkcd webcomic, in which participants have to reach a random location (chosen by a computer algorithm), prove their achievement by taking a picture of a Global Positioning System (GPS) receiver or another mobile device and then tell the story of their trip online. Proof based on non-electronic navigation is also acceptable.

Whereas other outdoor recreational activities like geocaching have a precise goal, geohashing is mainly fueled by its pointlessness, which is deemed funny by its players. The resulting geohashing community and culture is thus extremely tongue-in-cheek, supporting any kind of humorous behavior during the practice of geohashing and resulting in a parody of traditional outdoor activities. Navigating to a random point need not be pointless. Some geohashers document new mapping features they find on the OpenStreetMap project.

Source Wikipedia

Chapitre II

Consignes

- Seule cette page servira de référence : ne vous fiez pas aux bruits de couloir.
- Le sujet peut changer jusqu'à une heure avant le rendu.
- Si aucune information contraire n'est explicitement présente, vous devez partir du principe que les versions des langages et framework utilisés sont les suivantes (ou ultérieures) :
  - `Python 3`
  - `HTML5`
  - `CSS 3`
  - `Javascript EM6`
  - `Django 1.9`
  - `psycopg2 2.6`
- Sauf indication contraire dans le sujet, les fichiers en python de chaque exercice sur `Python` seul (d01, d02 et d03) doivent comporter à leur fin un bloc `if __name__ == '__main__'`: afin d'y insérer le point d'entrée dans le cas d'un programme, ou des tests dans le cas d'un module.
- Sauf indication contraire dans le sujet, chaque exercice des journées portant sur `Django` aura sa propre application dans le projet à rendre pour des raisons pédagogiques.
- Les exercices sont très précisément ordonnés du plus simple au plus complexe. En aucun cas nous ne porterons attention ni ne prendrons en compte un exercice complexe si un exercice plus simple n'est pas parfaitement réussi.
- Attention aux droits de vos fichiers et de vos répertoires.
- Vous devez suivre la procédure de rendu pour tous vos exercices : seul le travail présent sur votre dépot GIT sera évalué en soutenance.
- Vos exercices seront évalués par vos camarades de piscine.
- Vous ne devez laisser dans votre répertoire aucun autre fichier que ceux explicitement specifiés par les énoncés des exercices.
- Sauf indication contraire dans le sujet vous ne devez pas inclure dans votre rendu :

  - Les dossiers `__pycache__`.
  - Les éventuelles migrations.
    Attention, il vous est tout de même conseillé de rendre le fichier `migrations/__init__.py`, il n'est pas nécessaire mais simplifie la construction des migrations.
    Ne pas ajouter ce fichier n'invalidera pas votre rendu mais vous *devez* être capables de gérer vos migrations en correction dans ce cas.
  - Le dossier créé par la commande `collectstatic` de `manage.py` (avec pour chemin la valeur de la variable `STATIC_ROOT`).
  - Les fichier en bytecode Python (Les fichiers avec une extension en `.pyc`).
  - Les fichiers de base de donnée (notamment avec `sqlite`).
  - Tout fichier ou dossier devant ou pouvant être créé par le comportement normal du travail rendu.

  Il vous est recommandé de modifier votre `.gitignore` afin d'éviter les accidents.

- Lorsque vous devez obtenir une sortie précise dans vos programmes, il est bien entendu interdit d'afficher une sortie précalculée au lieu de réaliser l'exercice correctement.
- Vous avez une question ? Demandez à votre voisin de droite. Sinon, essayez avec votre voisin de gauche.
- Votre manuel de référence s'appelle `Google / man / Internet / ...`.
- Pensez à discuter sur le forum Piscine de votre Intra !
- Lisez attentivement les exemples. Ils pourraient bien requérir des choses qui ne sont pas autrement précisées dans le sujet...
- Par pitié, par Thor et par Odin ! Réfléchissez nom d'une pipe !

Chapitre III

Règles spécifiques de la journée

- Aucun code dans le scope global. Faites des fonctions !
- Chaque fichier rendu doit être terminé par un appel de fonction dans une condition identique à :

```
if __name__ == '__main__':
    your_function( whatever, parameter, is, required )
```

- Il est toléré de placer une gestion d'erreur dans cette même condition.
- Aucun import autorisé, à l'exception de ceux explicitement mentionnés dans la section 'Fonctions Autorisées' du cartouche de chaque exercice..
- L'interprete à utiliser est `python3`.

Chapitre IV

Exercice 00

	Exercice : 00
Exercice 00 : Antigravity	
Dossier de rendu : $ex00/$	
Fichiers à rendre : `geohashing.py`	
Fonctions Autorisées : `module sys et antigravity`	

Ceci est un petit exercice d'échauffement faisant écho au préambule de la journée. Rien de très compliqué.

Réalisez un petit programme nommé `geohashing.py` qui prend autant de paramètres que nécessaire au calcul d'un géohash typique et qui doit donc évidemment calculer ce géohash avant de l'afficher sur la sortie standard.

En cas d'erreur, le programme doit afficher un message pertinent que vous aurez choisi avant de quitter proprement.

Ce schéma peut, peut-être, vous aider : Geohashing algorithm

Chapitre V

Exercice 01

	Exercice : 01
Exercice 01 : Pip	
Dossier de rendu : $ex01/$	
Fichiers à rendre : `my_script.sh my_program.py`	
Fonctions Autorisées : `module path.py`	

`path.py` est une librairie qui implémente un objet Path autour du module `os.path` de Python, rendant son utilisation très intuitive.

Dans cet exercice, vous devez créer un script `bash` qui installe cette librairie, ainsi qu'un programme `Python` qui l'utilise.

Le script Shell doit répondre à cette description :

- Son nom doit avoir l'extension `.sh` car c'est un script `Shell`.
- Il doit afficher la version utilisée de pip.
- Il doit installer la version de développement de `path.py` depuis son repo `GitHub`, dans un dossier qui doit s'appeler `local_lib`, placé dans le dossier de rendu. Si la librairie a déjà été installé dans ce dossier, l'installation doit alors l'écraser.
- Il doit écrire les logs d'installation de `path.py` dans un fichier ayant pour extension `.log`
- Si la librairie a été correctement installée, il doit finalement exécuter le petit programme que vous devez également créer.

Le programme `Python` à créer est une composition de votre choix, qui doit néanmoins réspecter ces contraintes :

- Son extension doit être `.py` car c'est un programme `Python`.
- Il doit importer le module `path.py` depuis l'endroit où cette bibliothèque a été installée, grâce au script précédent.
- Il doit créer un dossier puis un fichier à l'interieur de ce dossier, écrire quelque chose dans ce fichier et enfin lire puis afficher son contenu.
- Il doit respecter les règles spécifiques de la journée.

Chapitre VI

Exercice 02

	Exercice : 02
Exercice 02 : requêter une API	
Dossier de rendu : $ex02/$	
Fichiers à rendre : `request_wikipedia.py requirement.txt`	
Fonctions Autorisées : `modules requests, json, dewiki et sys`	

Wikipédia est un formidable outil partagé que vous connaissez forcément. Il est disponible sur votre navigateur favori et même sous forme d'application mobile. Je vous propose de créer maintenant un outil qui vous permette de requêter ce site, aujourd'hui devenu indispensable, directement depuis votre terminal.

Pour ce faire, vous devez concevoir un programme nommé `request_wikipedia.py` qui prend en paramètre une `string` et effectue une recherche via l'API de Wikipédia avant d'écrire le résultat dans un fichier. Vous avez le choix de requêter l'API anglaise ou française.

- Le programme doit écrire un résultat, même si la requête est mal orthographiée. Prenez le site original pour exemple : si celui-ci vous trouve un résultat pour une requête donnée, alors votre programme aussi.
- Le résultat doit être débarrassé de tout formatage `JSON` ou `Wiki Markup` avant d'être écrit dans le fichier.
- Le nom du fichier doit avoir le format `nom_de_la_recherche.wiki` et ne doit contenir aucun espace.
- En cas d'absence de paramètres, de mauvais paramètres, de requête invalide, d'information non trouvée, de problème serveur, ou de tout autre problème : aucun ficher ne doit être crée et un message d'erreur pertinent doit être affiché sur la console.
- Incluez dans votre rendu un fichier `requirement.txt` qui sera utilisé en soutenance pour installer les librairies indispensables à votre programme dans un `VirtualEnv` ou sur le système.

La librairie dewiki n'est pas parfaite, nous ne cherchons pas le résultat le plus propre possible, ce n'est pas le but de cet exercice.

Lisez attentivement la documentation de l'API. Observez la structure de ce qui vous est renvoyé.

Voici un exemple de ce que nous attendons :

```
$>python3 request_wikipedia.py "chocolatine"
$>cat chocolatine.wiki



   Une chocolatine designe :
   * une viennoiserie au chocolat, aussi appelee pain au chocolat ou couque au chocolat ;
   * une viennoiserie a la creme patissiere et au chocolat, aussi appelee suisse ;
   * une sorte de bonbon au chocolat ;
   * un ouvrage d'Anna Rozen


   Malgre son usage ancien, le mot n'est entre dans le dictionnaire Petit Robert qu'en 2007 et dans le
        Petit Larousse qu'en 2011.

   L'utilisation du terme "Chocolatine" se retrouve egalement au Quebec, dont la langue a evolue a partir
         du vieux francais differemment du francais employe en Europe, mais cet usage ne prouve ni n'
        infirme aucune anteriorite, dependant du hasard de l'usage du premier commercant l'ayant introduit
         au Quebec.

    References



   Categorie:Patisserie
   Categorie:Chocolat
```

Chapitre VII

Exercice 03

	Exercice : 03
Exercice 03 : Parser du html	
Dossier de rendu : $ex03/$	
Fichiers à rendre : `roads_to_philosophy.py, requirement.txt`	
Fonctions Autorisées : `modules sys, requests et BeautifulSoup`	

La légende dit que : si vous partez d'un article quelconque de Wikipédia, que vous cliquez sur le premier lien dans l'introduction de cet article qui ne soit ni en italique ni entre parenthèse, puis que vous répetez ce processus en boucle, vous aboutirez invariablement sur l'article Philosophie.

Et bien sachez que ce n'est pas une légende ! (faites 'Oooooh !' avec un air abasourdi). En témoigne cet article de ... Wikipédia.

Cependant, comme vous ne croyez que ce que vous voyez de vos propres yeux, vous **devez** créer un programme qui met ce phénomène à l'épreuve en listant puis en comptant tous les articles entre votre requête et l'article Wikipédia : les **roads to philosophy**.

Ce programme doit se nommer `roads_to_philosophy.py` et avoir le comportement suivant :

- Le programme doit prendre en paramètre une string qui est un mot ou un ensemble de mots correspondant à une seule recherche Wikipédia.
- Le programme doit requêter une url du Wikipedia **anglais** identique à celle d'une recherche standard sur un navigateur. Autrement dit : il n'est **pas permis** d'utiliser l'API du site.

- Il doit parser la page `html` grâce à la librairie `BeautifulSoup` afin de :
  - Trouver l'éventuelle redirection, et la prendre en compte dans les **roads to philosophy**. Attention, il n'est pas question ici de redirection d'url.
  - Trouver le titre principale de cette page et l'ajouter aux **roads to philosophy**.
  - Trouver (s'il existe) **le premier lien du paragraphe d'introduction** conduisant à un autre article de Wikipédia. Plutôt que d'ignorer ce qui est en italique et entre parenthèse, le programme doit **ignorer** soigneusement les liens qui ne pointent pas vers un nouvel article, tels que ceux qui menant à la section aide de `Wikipédia`.
- Le programme doit recommencer a partir de l'**étape 2** en partant du lien obtenu a l'étape précédente jusqu'à tomber sur un des cas suivant :
  - Le lien mène à la page `philosophy`, auquel cas il doit imprimer sur la sortie standard les articles visités ainsi que le compte total de ces articles sous le format `<nombre> roads from <requête> to philosophy`
  - La page ne contenait aucun liens valides, le programme doit afficher : `It leads to a dead end !`.
  - Le lien conduit à une page déjà visitée, ce qui signifie que le programme s'apprete a boucler indéfiniement. Dans ce cas le message affiché doit être : `It leads to an infinite loop !`
- A ce stade, après avoir affiché sur la sortie standard les messages necessaires, le programme doit quitter proprement.

`Si à n'importe quel moment dans l'execution du programme, une erreur survient, telle qu'une erreur de connexion, de server, de paramètre, de requête ou autre : le programme doit quitter proprement avec un message d'erreur pertinent.`

Comme pour l'exercice précédent, vou devrez fournir avec votre programme un ficher `requirement.txt` afin de faciliter l'installation des librairies.

La sortie de votre programme doit ressembler à celle-ci :

```
$> python3 roads_to_philosophy.py "42 (number)"
42 (number)
Natural number
Mathematics
Ancient Greek
Greek language
Modern Greek
Colloquialism
Word
Linguistics
Science
Knowledge
Awareness
Conscious
Consciousness
Quality (philosophy)
Philosophy
17 roads from 42 (number) to philosophy !
$> python3 roads_to_philosophy.py Accuvio
It's a dead end !
$>
```

La communauté Wikipédia met régulièrement ses articles à jour. Il est possible qu'entre la date de rédaction de ce sujet et celle à laquelle vous effectuez cette formation, l'exemple accuvio ne soit plus une impasse.

Chapitre VIII

Exercice 04

	Exercice : 04
Exercice 04 : Virtualenv	
Dossier de rendu : $ex04/$	
Fichiers à rendre : `requirement.txt my_script.sh`	
Fonctions Autorisées : `tout`	

Demain est la première journée de la formation concernant le framework `Django`. Vous vous devez de préparer le terrain en configurant une petite installation facile.

Vous devez pour cela créer deux élements :

- Un fichier `requirement.txt` qui doit contenir les dernières versions stables de `django` et de `psycopg2`.
- Un script qui doit avoir ce comportement :
  - Avoir l'extension `.sh`
  - Créer un `virtualenv` sous `python3` nommé `django_venv`.
  - Installer le fichier `requirement.txt` que vous avez créé dans ce `VirtualEnv`.
  - En quittant, le `virtualenv` doit être **activé**.

Chapitre IX

Exercice 05

	Exercice : 05
Exercice 05 : Hello World	
Dossier de rendu : $ex05/$	
Fichiers à rendre : `tout fichier nécessaire`	
Fonctions Autorisées : `tout`	

Se contenter d'installer `Django` doit être frustrant, et c'est parfaitement compréhensible.

Vous allez donc achever cette journée sur les librairies en beauté en concevant votre premier `Hello World` avec `Django`.

Dans ce dernier exercice, vous devez suivre et adapter le tutoriel officiel afin de faire une page web qui, à partir de l'adresse `http://localhost:8000/helloworld`, affiche simplement le texte `Hello World !` sur votre navigateur.

Votre rendu doit être un dossier contenant le projet `Django`.