Cours 1: scripts python

1 - Préambule

Installer VSCode et python: suivre le guide: https://code.visualstudio.com/docs/python/python-tutorial (ignorer à partir de la section **Configure and run the debugger**).

—

Dans cette séance, nous allons travailler différentes notions:

1. Principe de fonctionnement d'un script python
2. Utilisation de la librairie python `argparse`, qui sert à parser les arguments passés à un script en ligne de commande.
3. Appels réseaux: un peu de théorie, et pratique en python.

2 - Exercices d'introduction

Example d'utilisation de la librairie argparse

```python
import argparse

parser = argparse.ArgumentParser(description='Argparse demo')
parser.add_argument("--name")
parser.add_argument("--year", type=int)
args = parser.parse_args()

print(f"Bonjour {args.name}, bienvenue en {args.year} !")
```

Exercice 1 : répétition de mots

Écrire un script python `repeat_words.py` tel que :

Entrée:

- un mot `mot`
- un nombre `n`

Sortie:

- affiche `n` fois le mot `mot` (séparés par une espace)

Exercice 2: Affichage de formes

Écrire un script python `affiche_forme.py` tel que :

Entrée:

- un choix: `triangle` ou `carré`
- un nombre `n`

Sortie:

- affiche un triangle ou un carré, de la taille demandée `n`

3 - Librairies python

Un grand nombre de librairies open-source sont publiées sur le site https://pypi.org/.

Pour installer une librairie, nous utiliserons le programme `pip` (voir aussi https://code.visualstudio.com/docs/python/python-tutorial#_install-and-use-packages).

> ⚠ **Attention**
>
> Pour s'assurer que le programme `pip` utilisé est bien celui auquel on s'attends, on utilisera la commande `which`:
>
> ```bash
> $ which pip
> /home/mmillet/dev/cours_algo/.venv/bin/pip
> ```

1. Installer les librairies `requests`, `python-dotenv`, ainsi que `ipython` et `devtools`

```bash
$ pip install requests python-dotenv ipython devtools
```

2. S'assurer qu'on la retrouve bien dans la liste des librairies installés:

```bash
$ pip list | grep requests
requests           2.32.3
```

3. Exécuter le programme `ipython`, et importer la librairie requests:

```Python
>>> import requests
```

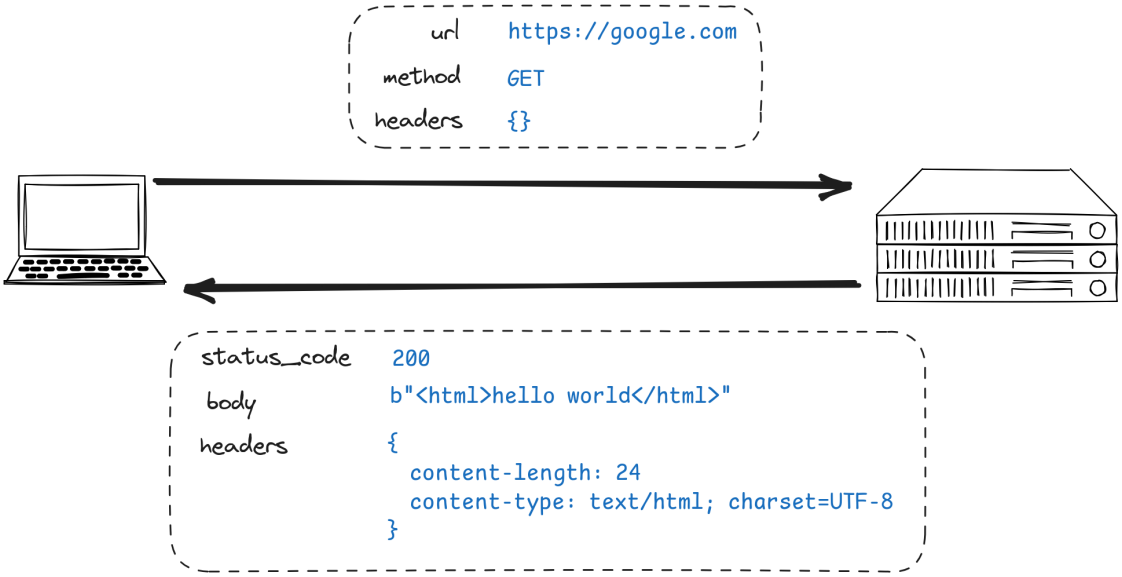
4. Lister des librairies python que vous connaissez, ainsi que ce à quoi elles servent.

5. Connaissez vous d'autres programmes qui servent à installer des librairies python ?

4 - Appels réseau http(s)

4.1 - Préambule

Voici un exemple simple de requête http:



◯ **Exercice 1 : compte de mot dans une page**

Entrée:

- une url `url`
- un mot `mot`

Sortie:

- le nombre de fois où `mot` apparaît dans la page à l'adresse `url`

◯ **Exercice 2 : temps moyen de chargement d'une page**

Entrée:

- une url `url`
- un entier `n`

Sortie:

- temps pour le 1er chargement de la page à l'adresse `url`
- temps moyen pour charger le contenu de la page à l'adresse `url` (sur `n` requêtes au total).

4.2 - Utilisation d'une API

Nous allons faire des exercices en utilisant l'API de https://github.com.

1. Créer un compte GitHub
2. Aller à l'adresse https://github.com/settings/tokens
3. Créer un token d'accès.
4. Stocker le token dans un fichier nommé `.env`, dans une ligne de la forme

```
GITHUB_TOKEN=ghp_...
```

Example d'utilisation de l'API GitHub

```python
import os
from dotenv import load_dotenv
import requests

load_dotenv()

token = os.environ["GITHUB_TOKEN"]
headers = {"Authorization": f"Bearer {token}"}
response = requests.get(
  "https://api.github.com/octocat",
  headers=headers
)

print("STATUS:", response.status_code)
print(response.text)
for key, value in response.headers.items():
    if "RateLimit" in key:
        print(f"{key}: {value}")
```

Exercice 3 : résumé compte GitHub

Entrée:

- un identifiant de compte GitHub `compte`

Sortie:

Pour chaque repository appartenant à `compte`, afficher:

- le nom du repository
- le nombre d'étoiles
- le nombre de branches
- le nombre de tags

On pourra par exemple lister les repositories du compte `django`

Exercice 4 : Création de gists

Entrée:

- une description
- un flag qui indique si le gist doit être public ou non
- une liste de noms de fichiers

Comportement:

Pour chaque nom de fichier:

- demander à l'utilisateur de fournir le contenu du fichier (fonction python `input`)
- créer un *gist* GitHub avec la description donnée, le statut public/privé, ainsi que les fichiers donnés