2N6 Programmation 2

avec python™

Les fonctions et modules
comment créer des modules

Utilisation de modules

```
# Fake Store REST API
import requests
import json
BASE_URL = 'https://fakestoreapi.com'

# faire une demande de tous les produits (GET)
response = requests.get(f"{BASE_URL}/products")
donne_req = respsonse.json()
print( json.dumps( donne_req, indent=4 ))
```

- Import des modules requests et json
- requests.get() utilise la fonction get() du module requests
- Le .get() nous retourne un objet objet "response" qui à la méthode .json() (provient de requests)
- json.dumps() utilise la fonction dumps() du module json

```
# Création de répertoires
import os

cours='420-2N6R'
groupe='1070'
os.makedirs(cours+'/'+groupe)
```

- Ici, on importe le module **os**.
- **os**.makedirs utilise la fonction makedirs() du module os

Modules importants

Un module regroupe des fonctions / objets / valeurs concernant un sujet commun.

- **os** : fonctions concernant le système d'exploitation
- **csv** : fonctions / classes pour traiter les données d'un fichier csv
- **requests** : fonctions pour faire des requêtes http
- **json** : fonctions pour traiter/créer des json.

- **sys** : fonctions/objet concernant l'interprétateur et le système d'exploitation
- **subprocess** : gestion/creation de processus et obtention de résultats
- **argparse :** permet de passer des arguments à un script par ligne de commande

Modules

- Il y a plusieurs façons de faire l'import, les deux premières sont équivalentes

```
/* test.py
1   import subprocess
2   import subprocess as sb
3   from subprocess import run
4   from subprocess import *
5
6   run(["echo","hello"],shell=True)
7
```

- Les syntaxes aux lignes 3 et 4 sont valides, mais on se retrouve à importer toutes les fonctions et variables dans notre namespace __main__

- La méthode run() vient de subprocess et si on import une fonction du même nom d'un autre module, les méthodes run() des deux modules entreront en conflit

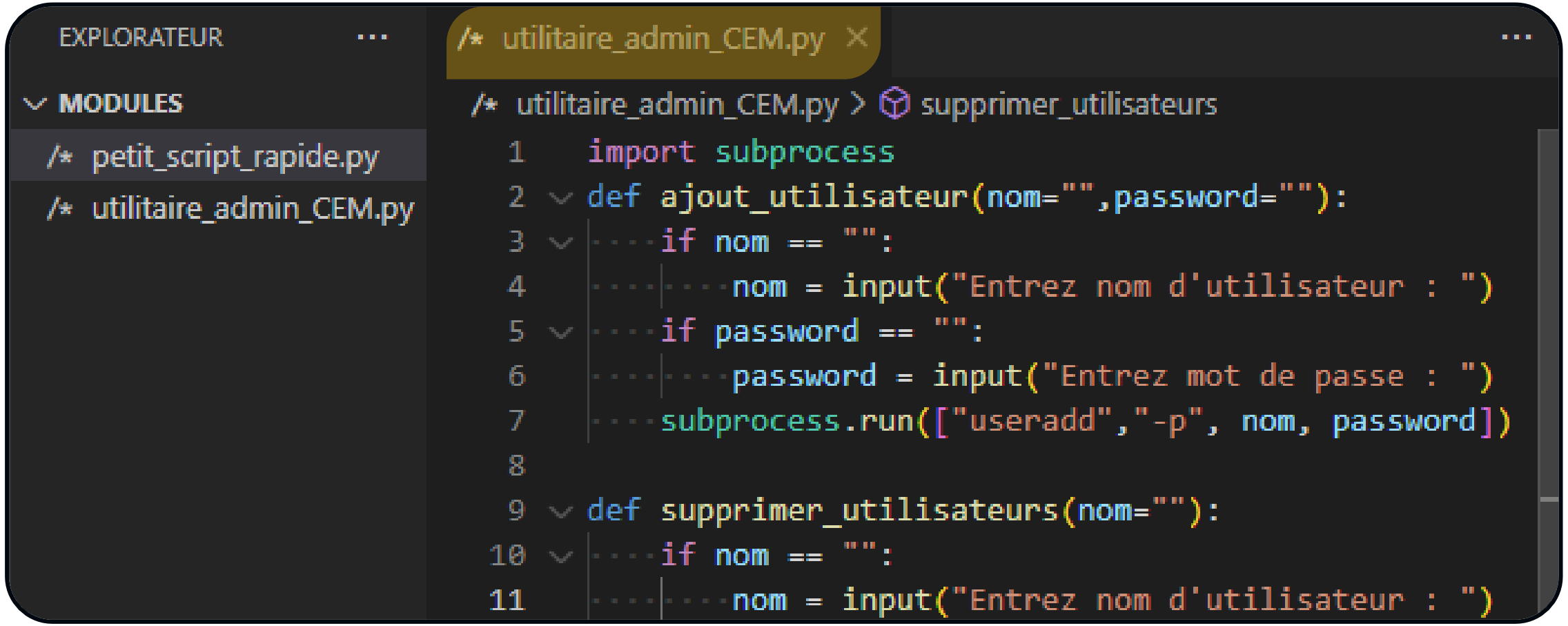
Modules

- Fichiers de code Python réutilisable dans différents programmes.
- Regroupe fonctions, variables et autres éléments ayant une fonctionnalité similaire.
- Code mieux organisé et plus facile à maintenir.
- Différentes parties du code séparées en modules distincts.
- Python possède la librairie standard qui regroupe de nombreux modules prédéfinis et prêts à l'emploi. Donc le code est déjà écrit et testé.

Création d'un module

- Si on réutilise le même code dans plusieurs scriptes, on va faire un nouveau module afin de s'assurer de toujours utiliser exactement le même code.
- Vous pouvez donc créer votre propre module pour regrouper des fonctions sur un sujet.
- Créer un nouveau module est aussi simple que d'écrire un script

Modules


EXPLORATEUR
MODULES
petit_script_rapide.py
utilitaire_admin_CEM.py
utilitaire_admin_CEM.py
utilitaire_admin_CEM.py > supprimer_utilisateurs
1 import subprocess
2 def ajout_utilisateur(nom="",password=""):
3 if nom == "":
4 nom = input("Entrez nom d'utilisateur : ")
5 if password == "":
6 password = input("Entrez mot de passe : ")
7 subprocess.run(["useradd","-p", nom, password])
8
9 def supprimer_utilisateurs(nom=""):
10 if nom == "":
11 nom = input("Entrez nom d'utilisateur : ")

Importer le module et utiliser ses fonctions

petit_script_rapide.py

```python
#importe notre module fait maison
import utilitaire_admin_CEM
import csv
#ouvre un csv et exécute une fonction provenant du module
with open("liste_nouveaux_utilisateurs.csv","r",) as fichier_users:
    csv_read = csv.reader(fichier_users)
    for ligne in fichier_users:
        nom = ligne[0]
        mot_de_passe = ligne[1]
        utilitaire_admin_CEM.ajout_utilisateur(nom, mot_de_passe)
```

Importer le module

Utiliser une de ses fonctions

Tester les cas limites

- Quand vous écrivez une fonction, vous devez tester les cas limites pour vérifier que tout est OK.
- Les cas limites sont les valeurs qui pourraient causer des problèmes.
- Par exemple, si votre fonction prend en paramètre un entier entre 1 et 10, les cas limites seraient:
  - Passer 1 en paramètre
  - Passer 10 en paramètre
  - Passer -1 en paramètre
  - Passer 11 en paramètre
  - Ne rien passer en paramètre
  - Passer 'patate' en paramètre

Nous allons voir plus de possibilités dans ces cas plus tard, en utilisant les try...except

Tester les cas limites pour toutes les fonctions de vos modules

- Quand vous écrivez un module, c'est très important de tester les cas limites des fonctions qui sont dans vos modules.
- On s'attend à ce qu'un module ait été bien testé et qu'il soit vraiment fonctionnel.
- Si une fonction pose problème, cette erreur existera dans tous les scripts utilisant ce module.

Un fichier : module ET script

- Lorsqu'on importe un module, l'interprétateur pas au travers de chacune des lignes du module.
- Toutes les lignes seront exécutées.
- Pour utiliser un même fichier en module et script, il faut inclure les lignes à exécuter en script qui doivent être incluses dans un bloc conditionnel :

```
if __name__ == "__main__":
    while True:
        var = input("Entrez le nomb
        if var.isdigit() :
```