Bibliothèques spécialisées

Scripting réseau

"Any fool can write code that a computer can understand. Good programmers write code that humans can understand."

Vinod Kumar Nair

1

Objectif

- Explorer les bibliothèques et concepts avancés pour l'administration système et réseau
- Maîtriser la manipulation de fichiers et l'interaction avec des services réseau
- Développer un code Python sécurisé
- Maîtriser les concepts de contrôle d'accès, d'authentification et de gestion des privilèges
- Appliquer des pratiques de protection des données et sécuriser les connexions réseau
- Implémenter des mécanismes de *logging* et d'audit sécurisés
- Acquérir les bases de la cryptographie en Python

2

AGENDA

1. **Format de données JSON**
   - Objet
   - Tableau
2. **Gestion des fichiers**
   - Modes d'ouverture
   - Fichiers texte
   - Fichiers JSON
3. **Services réseau et système**
   - Module sys
   - Module os
   - Module subprocess
   - Module psutil
   - Module socket
4. **Interagir avec des Services Web et API REST**
   - Module request
5. **Sécurisation du code**

3

Format de données JSON

2025-09-16

4

Format JSON

5

JSON (**J**ava**S**cript **O**bject **N**otation) est un format de données texte léger, conçu pour être facilement lisible par les humains tout en restant facile à analyser et à générer par les machines.

JSON est couramment utilisé pour échanger des données entre serveurs et applications web. Bien qu'il soit basé sur la syntaxe d'objet JavaScript, JSON est indépendant du langage de programmation et peut être utilisé avec presque tous les langages, y compris Python.

5

Format JSON

6

Structure des données en JSON

Les données JSON sont basées sur deux structures principales :

- Les **objets** sont délimités par des accolades **{ }** et contiennent des paires clé-valeur. Les valeurs sont accessibles en utilisant la clé.

6

Format JSON

7

Manipuler la structure des données en JSON

Ajouter un élément à l'objet

La fonction **item()** permet d'obtenir une vue de toutes les paires **clé-valeur** sous forme de tuples. Elle retourne une séquence de paires (clé, valeur), ce qui permet de parcourir facilement chaque élément d'un dictionnaire.

```
# Objet JSON initial
data = {
    "nom": "GONZALEZ",
    "age": 30,
    "ville": "Paris"
}

# Ajouter la nouvelle paire clé-valeur

data["email"] = "gonzalez@example.com"

# Afficher tous les éléments
for cle, valeur in data.items():
    print(f"{cle}: {valeur}")
```

7

Format JSON

8

Structure des données en JSON

Les données JSON sont basées sur deux structures principales :

- Les **tableaux** sont des listes de valeurs entourées par des crochets **[ ]**. En Python, ils correspondent aux listes.

```
[
    {
        "nom": "GONTIER",
        "age": 30
    },
    {
        "nom": "GONZALEZ",
        "age": 25
    }
]
```

8

Format JSON

9

Manipulation du **tableau** chaque élément du tableau est traité comme un dictionnaire avec des pairs clé-valeur

```python
import json
utilisateurs = [
        {
            "nom": "Martha",
            "age": 25
        },
        {
            "nom": "Inés",
            "age": 30
        }
    ]

# Ajouter un nouvel utilisateur
nouvel_utilisateur = {"nom": "Katy", "age": 35}
utilisateurs.append(nouvel_utilisateur)

# Afficher les données modifiées
for item in utilisateurs :
    print(f'Prénom : {item['nom']}, Age :{item['age']}')
```

9

Gestion de fichiers

2025-09-16

11

Manipulation de fichiers

12

Mécanismes de Base :

La manipulation de fichiers en Python se fait en trois étapes principales : ouvrir, opérer (lire/écrire) et fermer.

- La fonction **open()** : crée un objet fichier et établit la connexion entre le script et le fichier physique sur le disque.

Argument	Description	Exemple
Nom du Fichier	Le chemin ou le nom du fichier.	« exemple.txt"
Mode	Spécifie l'opération prévue (lecture, écriture, etc.).	'r' (Lecture), 'w' (Écriture, écrase), 'a' (Ajout/Append)

12

Manipulation de fichiers

Exemple :

```
# --- EXEMPLE DE LECTURE SANS 'WITH' ---

# Initialisation de l'objet fichier à None (bonne pratique de sécurité)
f = None
fichier_a_lire = "exemple_1.txt"

try:
    # 1. OUVERTURE : Spécification du mode 'r' (Lecture) et de l'encodage 'utf-8'.

# Si le fichier avait été encodé en 'latin-1', nous aurions dû utiliser 'latin-1' ici.
    f = open(fichier_a_lire, 'r', encoding='utf-8')

# Gestion des erreurs de base
except FileNotFoundError:
    print(f"Erreur : Le fichier '{fichier_a_lire}' n'a pas été trouvé.")
except UnicodeDecodeError:
    # Cette erreur survient si l'encodage 'utf-8' ne correspond pas au fichier réel
    print(
"Erreur : L'encodage 'utf-8' spécifié n'a pas pu décoder les octets du fichier.")

# 3. FERMETURE MANUELLE : Le bloc 'finally' garantit que f.close() sera exécuté.
finally:
    if f is not None:
        f.close()
        print("\nLe fichier a été fermé manuellement dans le bloc 'finally'.")
```

13

Manipulation de fichiers

14

Encodage :

La fonction ‘**open()**‘ en Python peut recevoir l'argument optionnel ‘**encoding**‘ pour spécifier la manière dont les données sont converties entre leur forme binaire et leur forme textuelle.

- **Lecture (Décoder) :** Python utilise l'encodage spécifié pour convertir les octets binaires du disque en caractères textuels lisibles.
- **Écriture (Encoder) :** Python utilise l'encodage pour convertir vos caractères textuels en octets binaires à enregistrer sur le disque.

Le Risque d'Oublier l'Encodage :

Si un fichier est lu avec le mauvais encodage, une erreur peut se produire ou obtiendrez des caractères étranges seront obtenus :

- **Erreur** ‘`UnicodeDecodeError`‘ : survient si l'encodage ne sait pas comment interpréter une séquence d'octets.
- **Caractères « Mojibake »** : Des caractères incorrects (souvent des carrés, points d'interrogation ou séquences comme ‘Ã © ‘ au lieu de ‘ é ‘).

14

Manipulation de fichiers

15

Encodage :

Encodage	Description	Utilisation
`'utf-8'`	L'encodage **standard moderne**. Il peut représenter n'importe quel caractère Unicode dans le monde (latin, cyrillique, chinois, emojis, etc.). Il est **fortement recommandé**.	Standard pour le Web et la plupart des applications.
`'latin-1'`	Un encodage plus ancien, couvrant la plupart des caractères d'Europe de l'Ouest (y compris les accents français). Moins flexible que ‘utf-8‘.	Fichiers historiques ou systèmes limités.
`'cp1252'`	Très similaire à ‘latin-1‘, c'est l'encodage traditionnel de Windows.	Systèmes Windows plus anciens.

15

Manipulation de fichiers

16

Opérations de lecture et écriture :

Une fois le fichier ouvert, l'objet fichier possède des méthodes pour interagir avec son contenu :

Opération	Description	Exemple
Écriture	‘write()‘	‘f.write("Nouvelle ligne")‘
Lecture	‘read()‘ (tout le contenu), ‘readline()‘ (une ligne), ‘readlines()‘ (toutes les lignes dans une liste)	‘contenu = f.read()‘
Fermeture	‘close()‘	‘f.close()‘

16

Manipulation de fichiers

Exemple :

```
f = None
fichier_a_lire = "exemple_1.txt"

try:
    # 1. OUVERTURE : Spécification du mode 'r' (Lecture) et de l'encodage 'utf-8'.
    # Si le fichier avait été encodé en 'latin-1', nous aurions dû utiliser 'latin-1' ici.
    f = open(fichier_a_lire, 'r', encoding='utf-8')

    # 2. OPÉRATION : Lecture de tout le contenu
    contenu = f.read()

    print("--- Contenu lu avec succès ---")
    print(contenu)

# Gestion des erreurs de base
except FileNotFoundError:
    print(f"Erreur : Le fichier '{fichier_a_lire}' n'a pas été trouvé.")
except UnicodeDecodeError:
    # Cette erreur survient si l'encodage 'utf-8' ne correspond pas au fichier réel
    print("Erreur : L'encodage 'utf-8' spécifié n'a pas pu décoder les octets du fichier."
)

# 3. FERMETURE MANUELLE : Le bloc 'finally' garantit que f.close() sera exécuté.
finally:
    if f is not None:
        f.close()
        print("\nLe fichier a été fermé manuellement dans le bloc 'finally'.")
```

17

Modes d'ouverture

18

Mode	Description
"r"	Lecture seule (le fichier doit exister)
"w"	Écriture seule (crée ou écrase le fichier)
"a"	Ajout seul (crée le fichier s'il n'existe pas)
"r+"	Lecture et écriture
"w+"	Écriture et lecture (écrase le fichier existant ou le crée)
"a+"	Ajout et lecture (ajoute en fin de fichier)
"rb"	Lecture en binaire
"wb"	Écriture en binaire
"ab"	Ajout en binaire

18

Manipulation de fichiers

19

Structure with :

Permet de gérer automatiquement les ressources, comme les fichiers. Lorsqu'on ouvre un fichier avec **with**, Python gère l'ouverture et la fermeture du fichier sans nécessiter d'appels supplémentaires, ce qui réduit les risques d'erreurs (comme oublier de fermer le fichier) et rend le code plus propre.

```
with open("nom du fichier", "mode", encoding="type encodage") as fichier:
    # opérations sur le fichier
```

Lecture seule "r"

Écriture seule "w"

Lecture et écriture "r+"

Encodage "utf-8", "latin-1"

19

Manipulation de fichiers

20

Le Rôle du Contexte sous l'Angle de la Sécurité (Risques Évités) :

L'utilisation de '**with**' offre une assurance optimale contre les risques liés à la mauvaise gestion des ressources, en particulier pour prévenir la **perte de données** et l'**épuisement des ressources système**.

Risque	Comment le Contexte `with` le Prévient
Perte de Données/Corruption	La fermeture ('close()') est l'action qui force l'écriture des données en attente dans la mémoire (**buffer**) vers le disque. En garantissant l'appel à `__exit__`, 'with' assure que cette écriture cruciale a lieu, même en cas de plantage (exception) dans le code.
Épuisement des Descripteurs de Fichier	Chaque fichier ouvert consomme une ressource système (descripteur). En garantissant la fermeture, 'with' **libère immédiatement** ce descripteur dès que le bloc est quitté, évitant que le programme ne dépasse la limite du système.
Oubli de Code	Il rend le code plus lisible et plus sûr, car il n'est plus nécessaire d'ajouter manuellement 'f.close()' ni de gérer les blocs 'try...finally' pour l'opération de nettoyage.

20

Modes d'ouverture

21

Exemple

```python
#Exemples modes d'ouverture
#
# Lecture seule
with open("exemple.txt", "r",encoding="utf-8") as fichier:
    contenu = fichier.read()
    print(f'Contenu du fichier {contenu}')

# Écriture seule, création si non existant
with open("exemple.txt", "w",encoding="utf-8") as fichier:
    fichier.write("Bonjour, monde !")

# Ajout de contenu à un fichier existant
with open("exemple.txt", "a", encoding="utf-8") as fichier:
    fichier.write("\nNouvelle ligne ajoutée.")

# Lecture et écriture
with open("exemple.txt", "r+",encoding="utf-8") as fichier:
    donnees = fichier.read()
    print(f'Contenu du fichier {donnees}')
    fichier.write("\nFin de l'exercice")
```

21

Modes d'ouverture

22

Exercice : Testez le script avec un fichier **exemple.text** contenant :

Exemple modes

Quel est le contenu du fichier à la fin du script ?

```python
#Exemples modes d'ouverture
#
# Lecture seule
with open("exemple.txt", "r",encoding="utf-8") as fichier:
    contenu = fichier.read()
    print(f'Contenu du fichier {contenu}')

# Écriture seule, création si non existant
with open("exemple.txt", "w",encoding="utf-8") as fichier:
    fichier.write("Bonjour, monde !")

# Ajout de contenu à un fichier existant
with open("exemple.txt", "a", encoding="utf-8") as fichier:
    fichier.write("\nNouvelle ligne ajoutée.")

# Lecture et écriture
with open("exemple.txt", "r+",encoding="utf-8") as fichier:
    donnees = fichier.read()
    print(f'Contenu du fichier {donnees}')
    fichier.write("\nFin de l'exercice")
```

22

Gestion de fichiers JSON

23

Python dispose du module intégré **json** pour manipuler des fichiers JSON, couramment utilisés pour échanger des données entre des systèmes.

Ce module fournit deux méthodes principales :pour travailler avec des fichiers JSON.

- **json.loads()**
- **json.dumps()**

Supposons un fichier **data.json** avec le contenu suivant :

```json
{
    "nom": "GONTIER",
    "age": 35,
    "ville": "Cachan"
}
```

23

Gestion de fichiers JSON

24

Lecture d'un fichier JSON

Pour lire le fichier **data.json** :

Permet de lire et de convertir le contenu d'un fichier JSON (chaîne json) en une structure de données Python, telle qu'un dictionnaire

```python
import json

with open("data.json", "r") as fichier:
    data = json.load(fichier)
    print(data)
```

24

Gestion de fichiers JSON

25

Écriture dans un fichier JSON

Pour écrire dans le fichier **data.json** :

Permet écrire des données Python (comme un dictionnaire ou une liste) dans un fichier au format JSON. Elle convertit les structures de données Python en texte JSON, puis les enregistre dans le fichier spécifié

```python
import json

data = {
    "nom": "GONZALEZ",
    "age": 30,
    "ville": "Paris"
}

with open("data.json", "w") as fichier:
    json.dump(data, fichier, indent=4)
```

25

Gestion de fichiers JSON

26

Ajouter des informations dans un fichier JSON

Pour ajouter dans le fichier **data.json** :

Il faut lire le contenu existant, le modifier en ajoutant une nouvelle clé ou élément, puis réécrire le tout dans le fichier

```python
import json

# Lecture du contenu existant
with open("data.json", "r") as fichier:
    data = json.load(fichier)

# Modification des données (ajout d'un nouvel é
lément)
data["email"] = "gonzalez@example.com"

# Réécriture des données avec l'élément ajouté
with open("data.json", "w") as fichier:
    json.dump(data, fichier, indent=4)
```

26

AGENDA

1 **Format de données JSON**
- Objet
- Tableau

27

2 **Gestion des fichiers**
- Modes d'ouverture
- Fichiers texte
- Fichiers JSON

3 **Services réseau et système**
- Module sys
- Module os
- Module subprocess
- Module psutil
- Module socket

4 **Interagir avec des Services Web et API REST**
- Module request

5 **Sécurisation du code**

27

Bibliothèques spécialisées

2025-09-16

28

Python et les nombres aléatoires

29

Pourquoi utiliser les **random** ?

- Générer des nombres aléatoires
- Simuler des tirages ou expériences
- Mélanger des séquences
- Créer des jeux et simulations

Bibliothèque **random** fonctions principales

- random() : renvoi un nombre aléatoire entre 0 et 1
- randint(a, b) : renvoi un entier aléatoire entre a et b
- choice(seq) : choisit un élément dans une séquence
- shuffle(seq) : mélange une liste en place

29

Exemple d'utilisation

30

Lances un dé

```
import random
resultat = random.randint(1, 6)
print(resultat)
```

30

Python et les nombres aléatoires

31

Exemple

- choice(seq) : choisit un élément dans une séquence

```
import random

# Liste de fruits
fruits = ["pomme", "banane", "orange", "kiwi",
"mangue"]

# Tirage aléatoire
fruit_choisi = random.choice(fruits)

print("Fruit choisi :", fruit_choisi)
```

31

Python et les nombres aléatoires

32

Exercice

- Ecrire un script Python qui génère un mot de passe qui respecte les recommandations de la CNIL. Le script demande à l'utilisateur la longueur souhaitée pour le mot de passe

32

Protection contre les attaques (DoS)

159

Les attaques par déni de service visent à rendre une ressource indisponible en l'inondant de requêtes. La protection contre ces attaques implique d'implémenter des stratégies telles que la limitation du taux de requêtes.

Il existe plusieurs solutions Python pour implémenter ce type de protection.

Exemple :

- **flask-limiter**
  - **Description** : Cette extension de Flask permet de limiter le nombre de requêtes qu'un client peut faire dans un intervalle de temps donné. Elle peut être utilisée pour prévenir les attaques par saturation du serveur en limitant le nombre de requêtes simultanées.

159

Protection contre les attaques (DoS)

160

Exemple : utilisation de la bibliothèque flask_limiter de flask

Syntaxe de la limitation : La chaîne de limitation "**5 per minute**" utilise un format spécifique pour définir le nombre de requêtes autorisées et la période de temps :

- **5** : Le nombre maximal de requêtes autorisées.
- **per** : L'unité de temps pour la limitation.
- **minute** : La période pendant laquelle les requêtes sont comptabilisées (autres valeurs possibles : second, minute, hour, day, etc.).

```python
from flask import Flask
from flask_limiter import Limiter
from flask_limiter.util import get_remote_address

app = Flask(__name__)
limiter = Limiter(get_remote_address, app=app)

@app.route("/api")
@limiter.limit("5 per minute")
def index():
    return "Hello, world!"

if __name__ == "__main__":
    app.run(debug=True)
```

160

Mises à jour et correctifs de sécurité

161

Les mises à jour régulières et les correctifs de sécurité sont essentiels pour maintenir un système sécurisé. Il est important de surveiller les vulnérabilités et d'appliquer des patches de sécurité dès qu'ils sont disponibles.

161