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

Bibliothèques spécialisées

2025-09-16

27

AGENDA

5 **Sécurisation du code**

- Gestion des entrées utilisateur
  - Les expressions régulières (**Regex**)
  - Bibliothèques spécialisées (**pydantic**, **cerberus**, **argparse**,..)
- Contrôle d'accès et authentification
- Chiffrement et gestion des secrets
- Protection contre les attaques par déni de service (DoS)
- Mises à jour et correctifs de sécurité

28

Les expressions régulières

29

Les **expressions régulières** (**Regex** abréviation de ***reg****ular* ***ex****pressions*) sont des séquences de caractères qui forment un modèle de recherche.

- Sont un moyen puissant de valider et de filtrer les entrées utilisateur (chaînes de caractères).
- Permettent de rechercher, valider ou extraire des patrons dans une chaîne de caractères. Il est également possible d'extraire des informations dans des chaînes.

Exemple :

```
modele = '^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$'
```


Les expressions régulières

30


```
modele = '^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$'
```


Syntaxe :

- Une expression régulière est généralement entourée de délimiteurs (des guillemets simples en Python) et peut contenir :
  - **Caractères littéraux** : **a**, **A**, **9**, ou **@** correspondent exactement à ces mêmes caractères dans le texte (la regex abc correspondra à la chaîne "abc").
  - **Métacaractères** : caractères spéciaux qui ont une signification particulière dans le contexte des regex :
    - **.** : Correspond à n'importe quel caractère sauf une nouvelle ligne.
    - **^** : Indique le début d'une ligne.
    - **$** : Indique la fin d'une ligne.
    - **** : Échappe un métacaractère pour qu'il soit traité comme un caractère littéral.

Les expressions régulières

31

```
modele = '^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$'
```

Syntaxe :

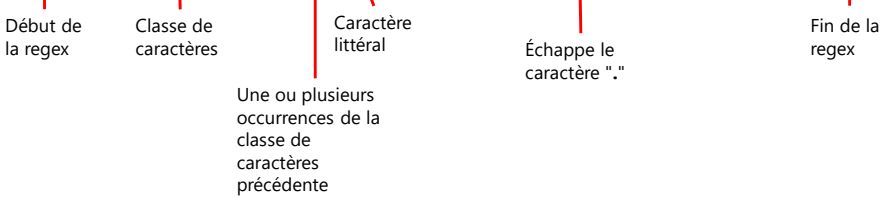
- **Classes de caractères** : permettent de définir un ensemble de caractères à faire correspondre. Par exemple :
  - **[abc]** : Correspond à a, b, ou c.
  - **[0-9]** : Correspond à n'importe quel chiffre de 0 à 9.
  - **[^abc]** : Correspond à tout caractère sauf a, b, ou c.
- **Quantificateurs** : spécifient combien de fois un élément peut apparaître.
  - ***** : Correspond à zéro ou plusieurs occurrences de l'élément précédent.
  - **+** : Correspond à une ou plusieurs occurrences de l'élément précédent.
  - **?** : Correspond à zéro ou une occurrence de l'élément précédent.

31

Les expressions régulières

32

```
modele = '^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$'
```



Exercice : identifiez le patron auquel correspond la regex

32

Les expressions régulières

33

Exemple : script python pour vérifier si une chaîne de caractères respect le modèle.

```
modele = '^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$'
```

Python fournit la bibliothèque **re** pour l'utilisation des expressions régulières, pour indiquer qu'une chaîne de caractères corresponde à une regex il faut :

```
modele = r'^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$'
```

33

Les expressions régulières

34

Exemple : script python pour vérifier si une chaîne de caractères respect le modèle.

```
import re

# Expression régulière pour valider un email
email_pattern = r'^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$'

email = input("Saisissez votre adresse email : ")

if re.match(email_pattern, email):
    print("Adresse email valide.")
else:
    print("Adresse email invalide.")
```

34

Les expressions régulières

35

Exercice :

- Décrire les règles qui doit respecter une adresse IP (IPv4)
- Écrire l'expression régulière pour valider une adresse IP (IpV4)

35

Les expressions régulières

36

Corrigé : écrivez l'expression régulière pour valider une adresse IP (IpV4)

```
pattern = r'^((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$'
```

36

Les expressions régulières

37

Exercice : écrivez le script Python pour vérifier si une chaîne de caractères corresponde à une adresse IP (IPv4).

Testez le script avec :

- 192.168.1.1
- 10.0.0.255
- 172.16.254.1
- 256.256.256.256
- 192.168.1.
- 192.168.1.01
- 0.0.0.0

37

Les expressions régulières

38

Corrigé : écrivez le script Python pour vérifier si une chaîne de caractères corresponde à une adresse IP (IPv4)

```python
import re

    # Vérifie si l'adresse IP est valide (IPv4)
    # La regex vérifie que chaque octet est compris entre 0 et 255.

# Expression régulière pour valider l'adresse IP
patron_ip = r'^((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$'

ip = input("Saisissez l'adresse IP : ")

if re.match(patron_ip, ip):
    print(f"Adresse valide : {ip}")
else:
    print(f"Adresse invalide : {ip}")
```

38

Les expressions régulières

39

Exercice : Modifiez le script ci-dessous pour indiquer à quelle classe appartiennent les adresses IP valides

```python
import re

    # Vérifie si l'adresse IP est valide (IPv4)
    # La regex vérifie que chaque octet est compris entre 0 et 255.

# Expression régulière pour valider l'adresse IP
patron_ip = r'^((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$'

ip = input("Saisissez l'adresse IP : ")

if re.match(patron_ip, ip):
    print(f"Adresse valide : {ip}")
else:
    print(f"Adresse invalide : {ip}")
```

39

Bibliothèques spécialisées

Visualisation de données

2025-10-06

40

Analyse de données

41

Ce module introduit des outils puissants du langage Python, comme **pandas** et **matplotlib**, qui permettent de transformer ces volumes de données en **analyses lisibles et exploitables**.

- Exploiter les logs pour extraire des informations clés
- Utiliser **pandas** pour : lecture, filtrage, regroupement
- Générer des graphes explicites avec **matplotlib**
- Identifier erreurs fréquentes, IP suspectes, bots
- Passer du traitement manuel à une analyse à grande échelle
- Communiquer les résultats via des visualisations exploitables

41

Bibliothèque matplotlib

42

Est une bibliothèque de visualisation en Python.

- Elle permet de créer des graphiques : courbes, barres, camemberts, etc.

Importation :

```
import matplotlib.pyplot as plt
```

Commandes de base :

- plt.plot() : pour tracer des courbes
- plt.scatter() : pour tracer des points
- plt.bar() : pour des diagrammes à barre
- plt.pie() : pour des camemberts
- plt.hist() : pour les histogrammes

42

Bibliothèque matplotlib

43

Exemple :

```python
# Importation du module pyplot de matplotlib et aliasé en tant que plt
# Ce module fournit des fonctions pour créer différents types de graphiques
import matplotlib.pyplot as plt

# Définition des points de données pour l'axe des x
x = [1, 2, 3, 4]
# Définition des points de données pour l'axe des y
y = [10, 20, 25, 30]

# Tracer les points de données (x, y) sous forme de graphique linéaire
plt.plot(x, y)
# Définir le titre du graphique
plt.title('Exemple de courbe')
# Définir le label pour l'axe des x
plt.xlabel('x')
# Définir le label pour l'axe des y
plt.ylabel('y')
# Activer la grille sur le graphique pour une meilleure lisibilité
plt.grid(True)
# Afficher la fenêtre du graphique
plt.show()
```

43

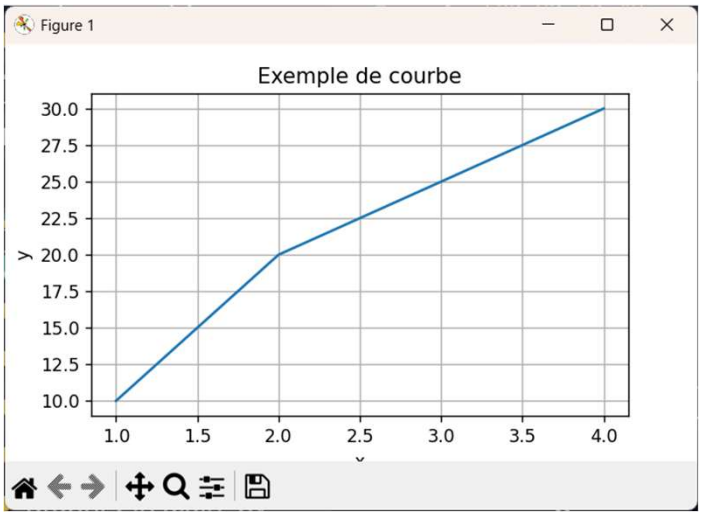
Bibliothèque matplotlib

44

Résultat :

Les points sont donnés sous la forma de listes

- plt.plot() : permet de tracer la courbe à partir des listes



44

Bibliothèque matplotlib

45

Exemple :

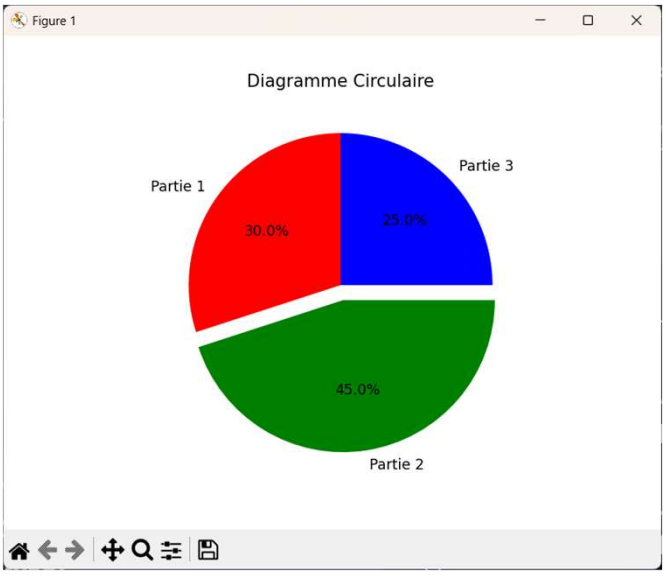
```
import matplotlib.pyplot as plt

# Exemple de données
labels = ['Partie 1', 'Partie 2', 'Partie 3']
sizes = [30, 45, 25]

# Création d'un diagramme circulaire
plt.pie(sizes, labels=labels, autopct='%1.1f%%',
startangle=90,
        colors=['red', 'green', 'blue'], explode=(0, 0.1
, 0))

# Personnalisation du graphique
plt.title('Diagramme Circulaire')

# Affichage du graphique
plt.show()
```

45

Bibliothèque matplotlib

46

Résultat :



46

Bibliothèques spécialisées

Traitement de données

2025-10-06

47

Bibliothèque argparse

48

Permet de gérer les arguments passés en ligne de commande à un script

Fonctionnalités

Fonction	Description
Définir des arguments	Permet de spécifier des options (--ip, --verbose, etc.)
Types de données	Automatiquement convertit les entrées (int, str, float, etc.)
Valeurs par défaut	Possibilité de définir une valeur si l'argument n'est pas précisé
Aide automatique	Génère l'aide avec -h ou --help
Validation	Gère les erreurs de type ou d'argument manquant

Exemple utilisation

```
python scanner.py --ip 192.168.1.1 --start-port 20 --end-port 80 --verbose
```

48

Bibliothèque argparse

49

Exemple

```python
import argparse

parser = argparse.ArgumentParser(description="Scanner de ports TCP")
parser.add_argument('--ip', required=True, help='Adresse IP à scanner')
parser.add_argument('--start-port', type=int, required=True, help='Port de début')
parser.add_argument('--end-port', type=int, required=True, help='Port de fin')
parser.add_argument('--verbose', action='store_true', help='Afficher les ports fermés')

args = parser.parse_args()

print(args.ip, args.start_port, args.end_port, args.verbose)
```

Exemple utilisation

python scanner.py --ip 192.168.1.1 --start-port 20 --end-port 80 --verbose

49

Bibliothèque pandas

50

Est une boîte à outils Python pour l'analyse et la manipulation de données tabulaires.

Elle est largement utilisée en data science, finance, et bien sûr en cybersécurité, notamment pour :

- analyser des logs (journaux d'événements, alertes),
- détecter des anomalies,
- agréger des statistiques,
- nettoyer et restructurer des données.

Pourquoi c'est utile en cybersécurité ?

- Pour analyser des logs massifs (authentification, firewall, système)
- Pour effectuer des recherches rapides sur les IP, utilisateurs, machines
- Pour grouper et visualiser des événements sur des périodes

50

Fonctions utiles pour l'analyse de logs

51

Fonction	Description
pd.read_csv()	Lire un fichier CSV (ou .txt) contenant des logs
df.head()	Afficher les premières lignes du DataFrame
df.info()	Afficher les types de colonnes et valeurs manquantes
df['colonne'].value_counts()	Compter les occurrences d'une valeur (IP, utilisateur…)
df['timestamp'] = pd.to_datetime()	Convertir une colonne date en objet datetime
df.groupby()	Grouper des données par utilisateur, date, source…
df[df['colonne'] == valeur]	Filtrer les lignes selon une condition
df.sort_values(by='colonne')	Trier les données
df.plot()	Graphe rapide depuis les données

51

Structures de données principales

52

- **DataFrame** : table à deux dimensions avec colonnes nommées.
- **Exemple** :

```
# --------- Exemple de DataFrame (table complète) ----------
# Un DataFrame est un tableau à deux dimensions avec colonnes nommées
data = {
    "Utilisateur": ["alice", "bob", "charlie"],
    "Action": ["Connexion", "Déconnexion", "Échec de connexion"],
    "Heure": ["2025-05-30 08:00", "2025-05-30 08:15", "2025-05-30 08:45"]
}
```

	Utilisateur	Action	Heure
0	alice	Connexion	2025-05-30 08:00
1	bob	Déconnexion	2025-05-30 08:15
2	charlie	Échec de connexion	2025-05-30 08:45

52

Pourquoi utiliser DataFrame plutôt que des structures natives comme *list*, *dict* ou *tuple* ?

53

Un ***DataFrame*** est tabulaire, comme une feuille Excel :
- ✓ colonnes nommées,
- ✓ index automatique ou personnalisé,
- ✓ accès facilité par nom de colonne.

- ✓ Des opérations vectorisées (rapides et optimisées),
- ✓ des fonctions de filtrage, regroupement, tri, fusion, jointure, nettoyage... sans boucle manuelle.

- ✓ Lecture/écriture facile depuis des fichiers CSV, Excel, SQL, JSON, etc.
- ✓ Gestion automatique des dates, chaînes, valeurs manquantes (NaN), encodage

- ✓ matplotlib / seaborn pour les graphiques,
- ✓ scikit-learn pour le *machine learning*,
- ✓ numpy pour le calcul numérique.

53

Structures de données principales

54

- **Series** : est une structure de données unidimensionnelle (comme une liste), fournie par la bibliothèque **pandas**. Elle associe des valeurs à des étiquettes d'index.

  Une ***Series*** est un hybride puissant qui combine le meilleur d'une liste (ordre, séquence), d'un dictionnaire (index personnalisé), avec des optimisations et fonctionnalités dédiées à l'analyse des données.

54

Structures de données principales

55

```python
import pandas as pd

# --------- Exemple de Series (colonne unique) ----------
# Une Series est comme une colonne, avec un index pour chaque valeur
series_exemple = pd.Series(["alice", "bob", "charlie"], name="
Utilisateurs")
print("Exemple de Series :")
print(series_exemple)
print("\nType :", type(series_exemple))
```

55

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 random
   - 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**

56

56

Protection contre les attaques (DoS)

176

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.

176

Protection contre les attaques (DoS)

177

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)
```

177

Mises à jour et correctifs de sécurité

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.

178