

PYTHON POUR LES RÉSEAUX

Cours complet organisé en 10 sections

Younes Derfoufi
Maître de conférences
CRMEF OUJDA

Public	Étudiants, enseignants, développeurs Python et débutants en administration réseau
Prérequis	Bases de Python : variables, conditions, boucles, fonctions, exceptions et modules
Objectif	Comprendre les réseaux et développer des scripts Python capables de communiquer, diagnos
Environnement	Python 3.x, terminal, réseau local ou machine de test

1. Fondamentaux des réseaux informatiques

1.1 Qu'est-ce qu'un réseau ?

Un réseau informatique relie des équipements afin qu'ils puissent échanger des données. Les principaux éléments sont les hôtes, interfaces réseau, commutateurs, routeurs et services. Internet est un ensemble mondial de réseaux interconnectés.

1.2 Modèles OSI et TCP/IP

Le modèle OSI comporte sept couches. La pile TCP/IP regroupe les fonctions autour de l'accès réseau, Internet, transport et application. Python intervient surtout aux couches transport et application.

1.3 Protocoles essentiels

IP assure l'adressage et l'acheminement. TCP fournit une communication fiable et orientée connexion. UDP est plus léger sans garantie de livraison. DNS traduit les noms de domaine en adresses IP. HTTP/HTTPS transporte les échanges Web.

1.4 Ports et client-serveur

Un port identifie un service sur une machine. Dans le modèle client-serveur, le serveur écoute sur une adresse et un port tandis que le client initie la connexion.

2. Python et les outils réseau

2.1 Modules utiles

La bibliothèque standard fournit `socket`, `ipaddress`, `urllib`, `http.client`, `ssl`, `selectors`, `threading`, `concurrent.futures` et `asyncio`. La bibliothèque `requests` simplifie les requêtes HTTP.

2.2 Identifier la machine locale

Le module `socket` permet d'obtenir le nom d'hôte et de résoudre des adresses.

```
import socket

nom = socket.gethostname()
print("Nom de la machine :", nom)
print("Adresse associée :", socket.gethostbyname(nom))
```

2.3 Résolution d'un nom

`getaddrinfo()` prend en charge IPv4 et IPv6.

```
import socket

infos = socket.getaddrinfo("python.org", 443)
for info in infos:
    print(info[0], info[1], info[4])
```

3. Adressage IPv4, IPv6 et sous-réseaux

3.1 Comprendre une adresse IP

IPv4 utilise 32 bits et IPv6 128 bits. Un préfixe CIDR, comme /24, indique la partie réseau.

3.2 Le module ipaddress

ipaddress permet de valider, comparer et manipuler des adresses et réseaux.

```
import ipaddress

ip = ipaddress.ip_address("192.168.1.10")
reseau = ipaddress.ip_network("192.168.1.0/24")
print(ip.version)
print(ip.is_private)
print(ip in reseau)
```

3.3 Parcourir les hôtes

hosts() génère les adresses hôtes utilisables d'un réseau.

```
import ipaddress

reseau = ipaddress.ip_network("192.168.10.0/29")
for hote in reseau.hosts():
    print(hote)
```

4. Programmation réseau avec les sockets TCP

4.1 Principe

Une socket est une extrémité de communication. Un serveur TCP utilise `bind()`, `listen()`, `accept()`, `sendall()` et `recv()`.

4.2 Serveur TCP

Exemple local sur le port 5000.

```
import socket

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as serveur:
    serveur.bind(("127.0.0.1", 5000))
    serveur.listen()
    connexion, adresse = serveur.accept()
    with connexion:
        donnees = connexion.recv(1024)
        connexion.sendall(b"Message reçu : " + donnees)
```

4.3 Client TCP

Le client se connecte au même port et échange des octets.

```
import socket

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client:
    client.connect(("127.0.0.1", 5000))
    client.sendall("Bonjour serveur".encode("utf-8"))
    print(client.recv(1024).decode("utf-8"))
```

5. Communication UDP

5.1 Caractéristiques

UDP transporte des datagrammes sans connexion et sans garantie de livraison ou d'ordre.

5.2 Récepteur UDP

`recvfrom()` retourne les données et l'adresse de l'émetteur.

```
import socket

with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as serveur:
    serveur.bind(("127.0.0.1", 6000))
    donnees, adresse = serveur.recvfrom(2048)
    print(adresse, donnees.decode("utf-8"))
```

5.3 Émetteur UDP

`sendto()` envoie directement un datagramme.

```
import socket

with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as client:
    client.sendto(b"Bonjour en UDP", ("127.0.0.1", 6000))
```

6. HTTP, HTTPS et API

6.1 HTTP

HTTP suit un modèle requête-réponse. Les méthodes courantes sont GET, POST, PUT, PATCH et DELETE. Les codes 2xx signalent généralement un succès, 4xx une erreur client et 5xx une erreur serveur.

6.2 urllib

`urllib.request` appartient à la bibliothèque standard.

```
from urllib.request import urlopen

with urlopen("https://www.python.org/", timeout=10) as reponse:
    print(reponse.status)
    print(reponse.read(200))
```

6.3 requests

`requests` facilite l'utilisation des API REST.

```
import requests

r = requests.get("https://httpbin.org/get", timeout=10)
r.raise_for_status()
print(r.status_code)
print(r.json())
```

6.4 HTTPS

HTTPS ajoute TLS afin de protéger confidentialité et intégrité et d'authentifier le serveur.

7. DNS avec Python

7.1 Rôle du DNS

Le DNS associe les noms de domaine à des informations réseau. A et AAAA concernent IPv4/IPv6, MX le courrier, CNAME les alias et TXT diverses informations.

7.2 Résolution

getaddrinfo() est général et compatible IPv4/IPv6.

```
import socket

domaine = "python.org"
for resultat in socket.getaddrinfo(domaine, 443):
    print(resultat[4])
```

7.3 Résolution inverse

gethostbyaddr() tente de retrouver un nom à partir d'une adresse IP.

```
import socket

try:
    print(socket.gethostbyaddr("8.8.8.8"))
except socket.herror as erreur:
    print("Résolution impossible :", erreur)
```

8. Diagnostic et automatisation réseau

8.1 Tester un service

Une méthode portable consiste à tenter une connexion TCP vers un service que l'on est autorisé à tester.

```
import socket

try:
    with socket.create_connection(("example.com", 443), timeout=3):
        print("Service accessible")
except OSError as erreur:
    print("Connexion impossible :", erreur)
```

8.2 Temps d'établissement

`time.perf_counter()` permet d'estimer le temps nécessaire à l'établissement d'une connexion.

```
import socket
import time

debut = time.perf_counter()
with socket.create_connection(("example.com", 443), timeout=5):
    duree = time.perf_counter() - debut
print(f"{duree:.3f} s")
```

8.3 Automatisation

Python peut vérifier des services, analyser des journaux, générer des rapports et appeler des API d'administration dans les limites des autorisations disponibles.

9. Transfert de données et concurrence

9.1 Transfert de fichiers

Un protocole applicatif doit préciser la taille et le format des données. `recv()` pouvant retourner moins d'octets que demandé, la réception doit être réalisée en boucle.

9.2 Plusieurs clients

`threading`, `concurrent.futures`, `selectors` et `asyncio` permettent de gérer plusieurs connexions.

```
import socket
from concurrent.futures import ThreadPoolExecutor

def traiter(connexion):
    with connexion:
        donnees = connexion.recv(1024)
        connexion.sendall(donnees.upper())

with socket.socket() as serveur:
    serveur.bind(("127.0.0.1", 7000))
    serveur.listen()
    with ThreadPoolExecutor(max_workers=5) as pool:
        while True:
            connexion, adresse = serveur.accept()
            pool.submit(traiter, connexion)
```

9.3 asyncio

`asyncio` repose sur une boucle d'événements et convient aux applications réalisant de nombreuses entrées-sorties réseau simultanées.

10. Sécurité réseau et projet final

10.1 Bonnes pratiques

Valider les entrées, imposer des timeouts, limiter les tailles reçues, traiter les exceptions et utiliser des protocoles chiffrés pour les données sensibles.

10.2 Secrets

Les mots de passe, jetons et clés d'API ne doivent pas être inscrits directement dans le code source.

10.3 Projet : moniteur de services

Le projet final vérifie une liste de services autorisés et produit un état synthétique.

```
import socket
from datetime import datetime

services = [("example.com", 443), ("python.org", 443)]
for hote, port in services:
    try:
        with socket.create_connection((hote, port), timeout=3):
            etat = "OK"
    except OSError:
        etat = "INDISPONIBLE"
    print(datetime.now().isoformat(timespec="seconds"), hote, port, etat)
```

10.4 À retenir

Python permet de manipuler les adresses, DNS, TCP, UDP, HTTP/HTTPS et d'automatiser de nombreuses tâches réseau. Il faut progresser sur des environnements contrôlés et respecter les autorisations.