Thème 2 : Langage et programmation

Leçon 1: Les APIs et leur utilisation en Python

1. Introduction aux APIs

1.1 Définition

Une API (Application Programming Interface) est un ensemble de règles et de protocoles qui permet à différentes applications de communiquer entre elles. C'est comme un contrat entre deux programmes : l'un propose des services, l'autre les utilise.

Analogie : Imaginez un restaurant. Le menu est l'API - il vous indique ce que vous pouvez commander sans avoir besoin de savoir comment le plat est préparé en cuisine. Le serveur fait le lien entre vous (le client) et la cuisine (le serveur).

1.2 Pourquoi utiliser des APIs ?

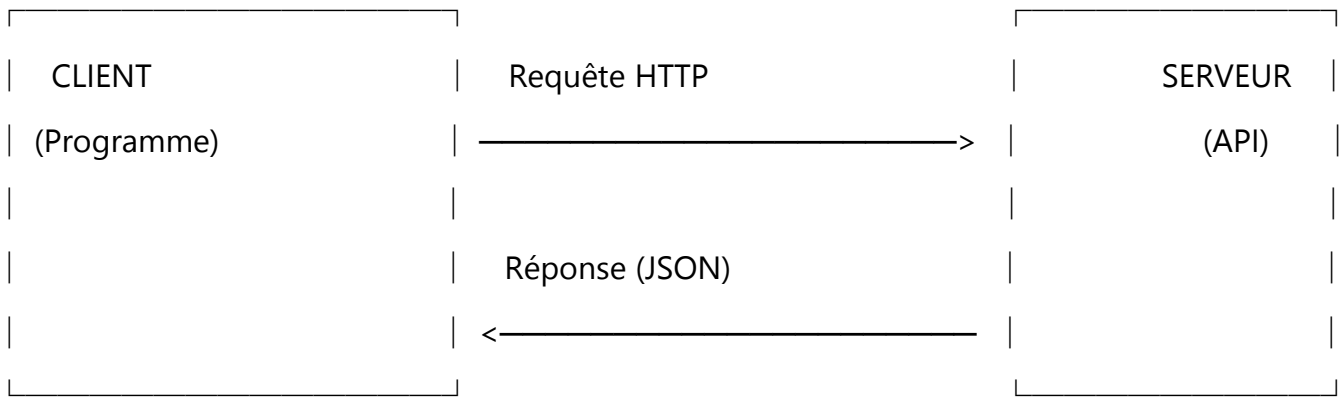
- ⇨ Réutilisation du code : Pas besoin de tout programmer soi-même
- ⇨ Accès à des services externes : Utiliser des fonctionnalités complexes (paiement, traduction, météo...)
- ⇨ Partage de données : Récupérer des informations mises à jour en temps réel
- ⇨ Interopérabilité : Faire collaborer différents systèmes

1.3 Exemples d'utilisation

Service	Fonctionnalité offerte
Google Maps API	Afficher des cartes, calculer des itinéraires
Stripe API	Gérer des paiements en ligne
OpenWeatherMap API	Obtenir des prévisions météorologiques
GitHub API	Accéder aux dépôts de code source
Discord/Telegram API	Créer des bots de messagerie

2. Architecture client-serveur des APIs Web

Les APIs modernes fonctionnent généralement sur le Web selon le principe suivant :



Processus :

1. Le client envoie une requête HTTP (GET, POST, PUT, DELETE...)
2. Le serveur traite la demande
3. Le serveur renvoie une réponse contenant les données (souvent en JSON)

3. Le format JSON

3.1 Présentation

JSON (JavaScript Object Notation) est le format standard pour échanger des données entre applications. Il est :

- Léger et lisible
- Indépendant du langage de programmation
- Structuré sous forme de paires clé-valeur

3.2 Syntaxe JSON

```
{
  "prenom": "Alice",
  "age": 17,
  "estEtudiant": true,
  "notes": [15, 18, 16, 19],
  "adresse": {
    "rue": "123 Avenue de la République",
    "ville": "Paris"
  },
  "diplome": null
}
```

Règles importantes :

- Les clés sont toujours entre guillemets doubles ""
- Types de valeurs possibles :
  - Chaînes de caractères : "texte"
  - Nombres : 42 ou 3.14
  - Booléens : true ou false
  - Tableaux : [1, 2, 3]
  - Objets : {"clé": "valeur"}
  - Null : null

3.3 Manipulation en Python

Le module json permet de travailler avec des fichiers JSON :

```
import json
```

```
# Lecture d'un fichier JSON
with open("donnees.json", "r") as fichier:
    data = json.load(fichier)
# data est maintenant un dictionnaire Python
print(data["prenom"])  # Affiche: Alice
print(data["notes"][0])  # Affiche: 15
# Écriture dans un fichier JSON
nouveau_data = {"nom": "Dupont", "score": 95}
with open("resultat.json", "w") as fichier:
    json.dump(nouveau_data, fichier, indent=2)
```

Correspondances JSON ↔ Python :

- {} → dict
- [] → list
- true/false → True/False
- null → None

4. Utiliser une API avec Python

4.1 Installation du module requests

Le module requests simplifie l'envoi de requêtes HTTP :

```
pip install requests
```

4.2 Requête GET simple

```
import requests
# Envoi d'une requête GET
url = "https://api.exemple.com/utilisateurs"
reponse = requests.get(url)
# Vérification du statut
if reponse.status_code == 200:
    # Conversion de la réponse JSON en dictionnaire
    donnees = reponse.json()
    print(donnees)
else:
    print(f"Erreur {reponse.status_code}")
```

Codes de statut HTTP courants :

- 200 : Succès
- 404 : Ressource non trouvée
- 500 : Erreur serveur

4.3 Requête avec paramètres

```
# Paramètres dans l'URL
parametres = {
    "ville": "Sfax",
    "unite": "metric"
}

reponse = requests.get("https://api.meteo.com/data", params=parametres)
donnees = reponse.json()
```

4.4 Requête POST

Utilisée pour envoyer des données au serveur :

python

```
# Données à envoyer
payload = {
    "texte": "Bonjour le monde!",
    "langue_source": "fr",
    "langue_cible": "en"
}

reponse = requests.post("https://api.traduction.com/translate", data=payload)
resultat = reponse.json()
print(resultat["traduction"])
```

5. Exemple pratique : API météo

5.1 Objectif

Créer un programme qui affiche la météo d'une ville donnée.

5.2 Code complet

```
import requests

def obtenir_meteo(ville, cle_api):
    """
    Récupère les informations météo pour une ville donnée
        Args:
        ville (str): Nom de la ville
        cle_api (str): Clé d'authentification API
         Returns:
        dict: Informations météo ou None en cas d'erreur
    """
    url = "https://api.openweathermap.org/data/2.5/weather"

    parametres = {
        "q": ville,
        "appid": cle_api,
        "units": "metric",
        "lang": "fr"
    }

    try:
        reponse = requests.get(url, params=parametres)
        reponse.raise_for_status()  # Lève une exception si erreur
        return reponse.json()
    except requests.exceptions.RequestException as e:
        print(f"Erreur lors de la requête : {e}")
        return None
def afficher_meteo(donnees):
    """Affiche les informations météo de manière formatée"""
    if donnees:
        ville = donnees["name"]
        temp = donnees["main"]["temp"]
        description = donnees["weather"][0]["description"]
        humidite = donnees["main"]["humidity"]

        print(f"\n🌤️  Météo à {ville}")
        print(f"   Température : {temp}°C")
        print(f"   Conditions : {description}")
        print(f"   Humidité : {humidite}%")
    else:
        print("Impossible d'afficher la météo")

# Programme principal
if __name__ == "__main__":
    CLE_API = "votre_cle_api_ici"
    ville_choisie = "Sfax"
```

```
meteo = obtenir_meteo(ville_choisie, CLE_API)
afficher_meteo(meteo)
```

6. Bonnes pratiques

6.1 Sécurité

❌ À éviter :

```
# Ne jamais écrire sa clé API directement dans le code
CLE_API = "abc123xyz789"
```

✅ À faire :

```
# Stocker la clé dans un fichier séparé ou variable d'environnement
import os
CLE_API = os.environ.get("API_KEY")
```

6.2 Gestion des erreurs

Toujours prévoir les cas d'échec :

```
try:
    reponse = requests.get(url, timeout=5)
    reponse.raise_for_status()
    donnees = reponse.json()
except requests.exceptions.Timeout:
    print("Le serveur met trop de temps à répondre")
except requests.exceptions.HTTPError as e:
    print(f"Erreur HTTP : {e}")
except requests.exceptions.RequestException as e:
    print(f"Erreur de connexion : {e}")
```

6.3 Respect des limites

La plupart des APIs imposent des limites :

- Nombre de requêtes par minute/jour
- Taille des données transférées

Solution : Mettre en cache les résultats pour éviter les requêtes répétées.

7. Exercices

Exercice 1 : API de blagues (facile)

Utilisez l'API https://official-joke-api.appspot.com/random_joke pour afficher une blague aléatoire en anglais.

Exercice 2 : Convertisseur de devises (moyen)

Créez un programme qui convertit un montant d'une devise à une autre en utilisant une API de taux de change (ex: ExchangeRate-API).

Exercice 3 : Recherche de livres (difficile)

Utilisez l'API d'Open Library pour créer un programme qui :

1. Recherche des livres par titre
2. Affiche les résultats (titre, auteur, année)
3. Permet de récupérer plus de détails sur un livre choisi

8. Synthèse

Concept	Description
API	Interface permettant la communication entre programmes
JSON	Format de données structuré et lisible
requests	Module Python pour faire des requêtes HTTP
GET	Récupérer des données
POST	Envoyer des données
Status codes	Codes indiquant le résultat de la requête (200, 404...)

Ressources complémentaires

- Documentation officielle de requests : https://requests.readthedocs.io/
- Tutoriel JSON : https://www.json.org/json-fr.html
- Liste d'APIs publiques : https://github.com/public-apis/public-apis
- Tester des APIs : https://reqbin.com/