TP Python — Générer une arborescence de projet

Contexte

Vous devez écrire un **script Python** permettant de créer automatiquement l'arborescence d'un projet à partir d'une **structure décrite par un dictionnaire**.

Exemple de structure :

```
STRUCTURE = {
    "README.md": "# TP\n",
    "src": {
        "main.py": 'print("Bonjour")\n',
    },
    "codes": {
        "prog1.py": "",
    }
}
```

```
📦 TP
├ 📜 README.md
├ 📂 src
│ └ 📜 main.py
└ 📂 codes
└ 📜 prog1.py
```

TP 1 — créer une arborescence simple

Objectifs

- Comprendre `Path`
- Manipuler un dictionnaire imbriqué
- Introduire la récursivité
- Créer fichiers et dossiers

Notions abordées

- `pathlib.Path`
- `dict`
- `isinstance`
- fonctions
- récursivité simple

Code (TP 1)

code_v1.py

```
from pathlib import Path

STRUCTURE = {
    "README.md": "# TP1 Python\n",
    "src": {
        "main.py": 'print("BTS CIEL - TP1")\n',
    },
}

def create_tree(base_path, structure):
    for name, content in structure.items():
        path = base_path / name

        # Si content est un dictionnaire → dossier
        # Sinon → fichier
        if isinstance(content, dict):
            path.mkdir(exist_ok=True)
            # Appel récursif pour parcourir les sous-dossiers
            create_tree(path, content)
        else:
            path.write_text(content, encoding="utf-8")

def main():
    root = Path("TP1_Python")
    root.mkdir(exist_ok=True)

    create_tree(root, STRUCTURE)
    print("Arborescence créée")

if __name__ == "__main__":
    main()
```

Exercices

- Exécuter le programme `code_v1.py` et vérifier le contenu du dossier créé
- Modifier le programme pour ajouter un dossier `tests` et vérifier le résultat
- Modifier le programme pour ajouter un fichier `__init__.py` et vérifier le résultat
- Modifier le contenu de `main.py`

TP 2 — Sécurité : vérifier avant d'écrire

Objectifs

- Tester l'existence d'un fichier ou dossier
- Éviter d'écraser par erreur
- Afficher des messages à l'utilisateur

Notions abordées

- `Path.exists()`
- conditions
- affichage (`print`)
- bonnes pratiques

Code (TP 2)

code_v2.py

```python
from pathlib import Path

STRUCTURE = {
    "README.md": "# TP2 Python\n",
    "src": {
        "main.py": 'print("BTS CIEL TP2")\n',
    },
}

def create_tree(base_path, structure):
    for name, content in structure.items():
        path = base_path / name

        if isinstance(content, dict):
            if path.exists():
                print(f"Dossier existant : {path}")
            else:
                path.mkdir()
                print(f"Dossier créé : {path}")
            create_tree(path, content)

        else:
            if path.exists():
                print(f"Fichier ignoré (existe déjà) : {path}")
            else:
                path.write_text(content, encoding="utf-8")
                print(f"Fichier créé : {path}")

def main():
    root = Path("TP2_Python")
    root.mkdir(exist_ok=True)

    create_tree(root, STRUCTURE)

if __name__ == "__main__":
    main()
```

Exercices

- Exécuter le programme `code_v2.py` et vérifier le contenu du dossier créé

- Ré-exécuter le programme `code_v2.py` et vérifier les messages affichés
- Ajouter un message différent si le dossier existe déjà

TP 3 — Projet Web

Objectifs

- Créer une structure pour un projet web

Structure du projet

```
📦 TP3_Web
 ├ 📂 pages
 │  ├ 📜 index.php
 │  ├ 📜 register.php
 │  ├ 📜 login.php
 │  └ 📜 logout.php
 ├ 📂 img
 │  └ 📜 logo.png
 ├ 📂 css
 │  └ 📜 style.css
 ├ 📜 config.php
 ├ 📜 bdd.sql
 └ 📜 cours_TP.md
```

Exercice

- Ecrire un programme Python pour créer la structure ci-dessus.
  - > *Vous pouvez réutiliser les codes (fonctions) écrits dans les TP précédents.*
  - > *Le fichier `logo.png` peut être créé vide (sans contenu).*

```
path.touch()
```

- Ajouter du contenu au fichier `config.php` (fichier qui permet de se connecter à la base de données).

> *Astuce : un fichier vide peut être représenté par une chaîne vide `""`.*

Exemple :

```
"logo.png": ""
```

TP 4 — Projet ESP32 / PlatformIO

Objectifs

- Créer une structure pour un projet ESP32

Structure du projet

```
MonProjet/
├─ src/
│  └─ main.cpp
├─ include/
├─ lib/
├─ test/
├─ platformio.ini
└─ .vscode/
```

- `platformio.ini` est **un fichier**
- `.vscode` est **un dossier**

Remarque : Certains dossiers peuvent être vides.

```
include/
lib/
test/
.vscode/
```

Exercice

- Ecrire un programme Python pour créer la structure ci-dessus.
- Ajouter du contenu au fichier `main.cpp` pour faire clignoter une LED branchée sur le pin de votre choix.

Exemple : `main.cpp`

```cpp
#include <Arduino.h>

void setup() {
    pinMode(2, OUTPUT);
}

void loop() {
    digitalWrite(2, HIGH);
    delay(500);
    digitalWrite(2, LOW);
    delay(500);
}
```

> Le pin 2 correspond souvent à la LED intégrée, mais cela dépend de la carte.

TP 5 — Créer un projet à partir d'un fichier JSON

Objectifs

- Comprendre le rôle d'un fichier **JSON**
- Lire un fichier avec Python
- Générer une arborescence à partir de données externes
- Séparer **données** / **programme**

Notions abordées

- format JSON
- module `json`
- `Path`
- dictionnaire
- fonctions

Contexte

En entreprise, la structure d'un projet est souvent décrite dans un **fichier de configuration**. Votre programme Python devra **lire un fichier JSON** et créer automatiquement l'arborescence correspondante.

Fichiers fournis

```
Données  → structure.json
Code     → tp5_json.py
```

`structure.json`

```
{
  "README.md": "# Projet JSON\n",
  "src": {
    "main.py": "print(\"Projet créé depuis un fichier JSON\")\n",
    "utils": {
      "helper.py": ""
    }
  },
  "tests": {}
}
```

Notes

- `{}` → dossier
- `"nom.ext": "texte"` → fichier texte
- `"nom.ext": ""` → fichier vide

Étape 1 — Vérifier le fichier JSON

- Créer le fichier `structure.json`
- Identifier dossiers / fichiers
- Modifier un nom de fichier

Étape 2 — Programme Python

tp5_json.py

```
from pathlib import Path
import json

def create_tree(base_path, structure):
    for name, content in structure.items():
        path = base_path / name

        if isinstance(content, dict):
            path.mkdir(exist_ok=True, parents=True)
            create_tree(path, content)
        else:
            path.write_text(content, encoding="utf-8")

def main():
    # Dossier racine du projet
    root = Path("Projet_JSON")
    root.mkdir(exist_ok=True)

    # Lecture du fichier JSON
    with open("structure.json", "r", encoding="utf-8") as f:
        structure = json.load(f)

    create_tree(root, structure)
    print("Projet créé à partir du fichier JSON")

if __name__ == "__main__":
    main()
```

Exercices

Exercice 1

Modifier le fichier `structure.json` pour ajouter :

- un dossier `docs`
- un fichier `docs/readme.txt`

Exercice 2

Modifier le programme pour afficher :

```
Création de : src/main.py
```

Indice :

```
print(f"Création de : {path}")
```

TP5 bis - Refaire le TP4 à partir d'un fichier JSON

Contexte

Lors du **TP4**, vous avez créé la structure d'un projet **ESP32 / PlatformIO** directement dans un dictionnaire Python.

Dans cet exercice, vous devez **refaire le TP4**, mais **en décrivant la structure du projet dans un fichier JSON**.

> Le programme Python devra lire le fichier JSON et créer automatiquement l'arborescence.

Travail demandé

1. Créer un fichier `structure_esp32.json`
2. Décrire dans ce fichier la structure suivante :

```
MonProjet/
├─ src/
│  └─ main.cpp
├─ include/
├─ lib/
├─ test/
├─ platformio.ini
└─ .vscode/
```

3. Ajouter du contenu au fichier `main.cpp`
4. Modifier votre programme Python pour lire ce fichier JSON

Exemple (corrigé partiel)

> *Vous pouvez vous inspirer de l'exemple ci-dessous.*

`structure_esp32.json` (exemple)

```
{
  "src": {
    "main.cpp": "#include <Arduino.h>\n\nvoid setup() {\n}\n\nvoid loop()
```

```
{\n}\n"
  },
  "include": {},
  "lib": {},
  "test": {},
  ".vscode": {},
  "platformio.ini": "[env:esp32dev]\nplatform = espressif32\nboard =
esp32dev\nframework = arduino\n"
}
```

Rappel :

- `{}` → dossier vide
- `"nom.ext": "texte"` → fichier texte

TP 6 — Publier le projet PlatformIO (TP4) sur Github

> Une fois votre projet généré (TP4 ou TP5 bis), vous devez le publier sur GitHub.

Objectifs

Mettre en ligne le projet généré afin de :

- sauvegarder le travail
- partager le code
- utiliser un workflow standard

Pré-requis

- Un compte GitHub
- Git installé sur votre machine
- Le projet (TP4) a été généré avec succès

Travail demandé : Publier votre projet sur GitHub **sans fichier `.gitignore`**

1. Créer un dépôt **vide** sur GitHub (via l'interface web)
    - Connectez-vous sur https://github.com
    - Cliquez sur **New repository**
    - Nom du dépôt : identique au dossier du projet
    - Ne cochez **aucune option** (pas de README, pas de licence)
    - Créez le dépôt
2. Dans le dossier du projet généré :

```
# Initialiser le dépôt Git en local
git init
git status
```

```
# Ajouter et valider les fichiers
git add .
git commit -m "Initialisation de la structure du projet"
# git log --oneline (optionnel)
```

3. Lier le dépôt local au dépôt GitHub :

```
git branch -M main
# Remplacer <user> et <repo> par vos informations
git remote add origin https://github.com/<user>/<repo>.git
git push -u origin main
# GitHub demandera un **token** si HTTPS est utilisé.
git status
```

Résultat

- Le projet est visible sur GitHub
- Tous les fichiers générés apparaissent dans le dépôt
- Le dépôt contient **au moins un commit**
- Constater sur GitHub la présence de fichiers indésirables (`.pio/`, `.vscode/` etc.)

`.gitignore`

1. Ajouter un fichier `.gitignore` adapté à PlatformIO
2. Constater que :
    - les fichiers déjà commités **restent visibles**
3. Les supprimer correctement du dépôt Git

> Car `.gitignore` empêche le *suivi futur*, il ne supprime **pas** ce qui est déjà suivi. **“.gitignore ne nettoie pas le passé.”**

4. Retirer les fichiers du suivi Git (sans les supprimer du disque)

```
git rm -r --cached .pio
git rm -r --cached .vscode
# Ou avec une seule commande :
# git rm -r --cached .pio .vscode
```

5. Commit de correction

```
git add .gitignore
git commit -m "Ajout de .gitignore et suppression des fichiers
ignorés"
git push
```

Résultat

- Le dépôt GitHub ne contient plus de fichiers inutiles
- `.gitignore` est présent
- Un nouveau commit corrige l'erreur
- Dossier toujours présent localement
