Spécial révision Python

Séance 1 Bases et lecture guidée d'un code complet

Précieux S. AMOUSSOU

Décembre 2025

Club MIATeX

Plan de la séance

1. Partie I Généralités et rappels essentiels. *(35 min)*
2. Partie II Code Python complet (construction pas à pas). *(55 min)*
3. Activité pratique et mini-quiz. *(10 min)*

Partie I Généralités

But de la partie I

- Donner ou rappeler les notions minimales nécessaires pour comprendre le code.
- Uniformiser les connaissances A1/A2 : A1 a la base, A2 gagne en précision.
- Préparer à programmer en direct et à interpréter les erreurs.

Langage interprété vs compilé 3 phrases

- **Interprété** : le code est lu/exécuté ligne à ligne par l'interpréteur (Python).
- **Compilé** : on traduit tout le programme en code machine avant exécution.
- Python est interprété : cycle rapide d'édition / exécution (idéal pour apprentissage).

Rappels syntaxe style (essentiels)

- Indentation obligatoire (4 espaces recommandés).
- Commentaires : `#...` ; blocs docs : triple quotes.
- Convention noms : `snake_case` pour fonctions/variables, `CamelCase` pour classes.
- Exécution interactive : REPL / notebooks utiles pour tests rapides.

Types de base (exemples rapides)

- `int` : entiers (ex : `42`)
- `float` : réels (ex : `3.14`)
- `str` : chaînes (ex : `"salut"`)
- `bool` : booléens (`True`/`False`)

- Conversions : `int(x)`, `float(x)`, `str(x)`
- Entrée utilisateur : `input("Invite: ")` $\rightarrow$ string
- Affichage : `print(...)`

Structures de contrôle rappel express

- Condition : `if / elif / else`
- Boucles : `for ... in ...` (itérable), `while ...`
- Interrompre : `break`, `continue`

Astuce pédagogique
Montrer un `for i in range(5):` et l'exécuter en direct pour vérifier la compréhension.

Collections essentielles

- `list` : mutable, ordonnée `append()`, `pop()`
- `tuple` : immuable, ordonné
- `dict` : paires clé valeur parcours par clés ou items
- Itération et compréhension de listes (list comprehensions)

Fonctions et classes notion rapide

- **Fonction** : `def nom(args): return ...` encapsule du comportement.
- **Classe** : `class Nom: def __init__(...): ...` encapsule état + méthodes.
- Pour la séance : implémentation simple d'une classe `Etudiant`.

Partie II Code guidé

Approche pédagogique pour la partie code

- Présenter un **programme complet** couvrant les notions vues.
- À chaque étape : expliquer l'algorithme (pseudo-code) puis taper/faire exécuter le code Python.
- Les participants programment et testent au fur et à mesure.

But du programme

Mini-application **Gestion d'étudiants** : création d'étudiants, ajout de notes, calcul de moyenne, affichage et classement. Montre variables, boucles, fonctions, collections, classes.

Code Python (complet

```
# gestion_etudiants.py
class Etudiant:
    def __init__(self, nom)
        :
         self.nom = nom
         self.notes = []
```

Pseudo-code / Algorithme

1. Définir la structure Etudiant (nom, liste de notes) 2. Fonctions : créer étudiant, afficher menu 3. Boucle principale : - afficher menu - lire choix - selon choix : créer, ajouter note, affi-

Code (variables / input)

```
def creer_etudiant():
    nom = input("Nom
        étudiant : ").strip
        ()
    return Etudiant(nom)
```

Algorithme (section)

- Demander une chaîne (nom) - Nettoyer espaces (strip) - Créer instance Etudiant avec ce nom - Retourner l'objet

- **Points pédagogiques** : conversion (si on demandait un âge), validation d'entrée, usage de `strip()`.
- **Erreurs fréquentes** : aucun nom saisi gérer cas vide.

Code (ajout note)

```
def ajouter_note_classe(
    classe):
    nom = input("Nom
        étudiant : ").strip
        ()
    note = float(input("
        Note (0-20) : ").
        strip())
    for et in classe:
        if et.nom == nom:
            et.ajouter_note
                (note)
            print("Note
```

Algorithme (section)

- Lire nom et note - Convertir note en float - Parcourir liste 'classe' : - si nom trouvé ajouter note et sortir - Si non trouvé message d'erreur

Code (moyenne afficher)

```
def afficher_moyennes(
    classe):
    for et in classe:
        print(f"{et.nom} :
            moyenne = {et.
            moyenne():.2f}")
```

Algorithme (section)

- Parcourir chaque étudiant - Appeler sa méthode moyenne() - Afficher formaté (2 décimales)

Astuce (A2) : séparer logique (calcul) et I/O (affichage) facilite tests unitaires.

Code (classe Etudiant)

```
class Etudiant:
    def __init__(self, nom):
        self.nom = nom
        self.notes = []

    def ajouter_note(self, note):
        self.notes.append(note)

    def moyenne(self):
        return sum(self.
```

Algorithme (section)

- Attributs : nom, liste de notes - Méthode ajouter$_{n}ote$: $pushdanslaliste$ — $Mthodemoyenne$: $somme/nombre, grerlistevide$

Section 5 Classement (extrait)

Code (classement)

```
def classement(classe):
    classe_triee = sorted(
        classe, key=lambda e
        : e.moyenne(),
        reverse=True)
    for rang, et in
        enumerate(
        classe_triee, start
        =1):
         print(f"{rang}. {et
             .nom} - {et.
             moyenne():.2f}")
```

Algorithme (section)

- Trier la liste par moyenne décroissante - Énumérer et afficher rang, nom, moyenne

Exécution guidée en classe

- **Étape 1 :** Copier le squelette minimal et exécuter (vérifier imports, indentation).
- **Étape 2 :** Ajouter la classe `Etudiant`, tester méthodes indépendamment.
- **Étape 3 :** Implémenter ajout des notes et afficher moyennes.
- **Étape 4 :** Ajouter gestion d'erreurs et améliorations (validation, messages).

Remarque pratique
Encourager l'utilisation d'un IDE simple (VSCode, Thonny) ou d'un notebook pour tests rapides.

Activité pratique

1. Formez des paires / petits groupes.
2. Implémentez le programme pas à pas (squelettes fournis).
3. Tests obligatoires : ajoutez 3 étudiants, 2 notes chacun, afficher classement.
4. Bonus (si temps) : sauvegarder/charger la classe depuis un fichier JSON.

Critères d'évaluation rapide :

- code qui s'exécute sans crash ;
- fonctions séparées ;
- comportement attendu (moyennes correctes).

Mini-quiz (10 min)

1. Comment vérifier si une liste est vide ? (A1)
2. Quelle instruction permet d'interrompre une boucle `for` ? (A1)
3. Pourquoi utiliser une méthode `moyenne()` dans la classe ? (A2)
4. Donnez un cas où `try/except` est utile dans l'exemple. (A2)

Ressources approfondissements

- Documentation Python officielle : `https://docs.python.org/fr/3/`
- Tutoriaux courts : OpenClassrooms, Real Python
- Exercices pratiques : repl.it / Replit, HackerRank, LeetCode (niveau débutant)
- Pour A2 : tests unitaires (module `unittest`), bonnes pratiques (PEP8)

Conclusion et suites

- Cette séance propose un **survol pratique et guidé** de Python.
- Séance suivante : étude d'une bibliothèque + petits projets (partie 2).

Merci pour votre attention !

Contact (Whatsapp) : 0160406083