**Session 1 : Rappels et Bonnes Pratiques en Python **

Objectif : Comprendre les principes fondamentaux de Python et adopter les bonnes pratiques

1. Introduction (15 min)

Pourquoi Python ?

- Langage interprété et dynamique
- Syntaxe simple et lisible
- Polyvalent : science des données, développement web, automatisation

Rappel des concepts de base

Types de base

```
# Entiers et flottants
x = 42
y = 3.14
print(type(x), type(y))  # <class 'int'> <class 'float'>

# Chaînes de caractères
s = "Bonjour, monde !"
print(s.upper())  # BONJOUR, MONDE !

# Booléens
flag = True
print(not flag)  # False
```

Structures de contrôle

```
# Conditionnelles
x = 10
if x > 5:
    print("x est grand")
else:
    print("x est petit")

# Boucles
for i in range(5):
    print(i)
```

Fonctions

```
def carre(n):
    return n * n

print(carre(5))  # 25
```

---

2. Manipulation avancée des structures de données (1h30)

Listes : Concepts sous-jacents et optimisations

- **Mutabilité :** Les listes sont modifiables, ce qui permet d'économiser de la mémoire.
- **Allocation dynamique :** Python optimise les performances en pré-allouant de l'espace supplémentaire.

Exemples de manipulation avancée

```
# Création et modification
liste = [1, 2, 3, 4]
liste.append(5)
print(liste)  # [1, 2, 3, 4, 5]

# Accès par indices négatifs
print(liste[-1])  # 5
```

Compréhensions de liste : Gains de performance

```
# Boucle classique
squares = []
for i in range(10):
    squares.append(i ** 2)

# Équivalent avec compréhension de liste
squares = [i ** 2 for i in range(10)]
print(squares)
```

Dictionnaires : Structures associatives performantes

- **Concept clé :** accès en O(1) grâce aux tables de hachage.
- **Utilisation optimale :** lorsque les données ont des associations clé-valeur.

Exemples pratiques

```python
etudiants = {"Alice": 18, "Bob": 20}
print(etudiants["Alice"])  # 18

# Ajout d'une entrée
etudiants["Charlie"] = 22

# Parcours des éléments
for nom, age in etudiants.items():
    print(f"{nom} a {age} ans")
```

Tuples et Ensembles : Structures immuables et non redondantes

Tuples : Optimisation mémoire et sécurité

```python
coordonnees = (10, 20)
x, y = coordonnees  # Décomposition
print(x, y)  # 10 20
```

Ensembles : Suppression automatique des doublons

```python
nombres_uniques = {1, 2, 2, 3, 4, 4, 5}
print(nombres_uniques)  # {1, 2, 3, 4, 5}
```

---

3. Bonnes pratiques de code (30 min)

Convention de nommage et lisibilité (PEP8)

- **Variables et fonctions :** `snake_case`
- **Classes :** `CamelCase`

```python
def aire_du_cercle(rayon):
    return 3.14 * rayon ** 2
```

Documentation et modularité

Docstrings pour clarifier l'intention du code

```python
def somme(a: int, b: int) -> int:
    """Retourne la somme de deux nombres entiers."""
    return a + b
```

Modularisation avec des fichiers séparés

- Séparer les fonctions dans des fichiers dédiés pour la réutilisabilité.

---

4. Exercices pratiques (45 min)

1. **Créer un dictionnaire des carrés de nombres**

```
def carres(nombres):
    return {n: n**2 for n in nombres}

print(carres([1, 2, 3, 4]))
```

2. **Éliminer les doublons d'une liste avec un ensemble**

```
def supprimer_doublons(liste):
    return list(set(liste))

print(supprimer_doublons([1, 2, 2, 3, 4, 4, 5]))
```

3. **Demander des informations à l'utilisateur et les stocker**

```
utilisateur = {}
utilisateur["nom"] = input("Entrez votre nom : ")
utilisateur["âge"] = int(input("Entrez votre âge : "))
print(utilisateur)
```

Creer un programme qui en partant d'une collection quelconque renvoie les éléments un à un à l'utilisateur à la suite d'un input dans le terminal en utilisant les génératrices