JOUR 1_4

Les collections

Les collections sont des objets Python qui permettent de stocker et de manipuler des données.

Les collections sont utilisées pour :

- **Stocker des données** : les collections permettent de stocker des données de manière efficace.
- **Manipuler des données** : les collections permettent de manipuler des données de manière concise et efficace.

Types de collections

Python fournit de nombreux types de collections, dont :

- **Les listes** : les listes sont des collections ordonnées qui peuvent contenir des éléments de n'importe quel type.
- **Les tuples** : les tuples sont des collections ordonnées qui sont immuables.
- **Les dictionnaires** : les dictionnaires sont des collections associatives qui associent des clés à des valeurs.
- **Les ensembles** : les ensembles sont des collections non ordonnées qui ne contiennent pas d'éléments dupliqués.

Listes

Les listes sont des collections ordonnées qui peuvent contenir des éléments de n'importe quel type.

Les listes sont définies en utilisant le symbole `[]`. Les éléments d'une liste sont séparés par des virgules.

Exemple

```
# Indexation avancée avec loc et iloc
liste = [1, 2, 3, 4, 5]
```

La liste `liste` contient les éléments `1`, `2`, `3`, `4`, et `5`.

Opérations sur les listes

Les listes supportent un large éventail d'opérations, dont :

- **L'ajout d'éléments** : les listes permettent d'ajouter des éléments à la fin de la liste.
- **La suppression d'éléments** : les listes permettent de supprimer des éléments de la liste.
- **La recherche d'éléments** : les listes permettent de rechercher des éléments dans la liste.
- **La modification d'éléments** : les listes permettent de modifier des éléments dans la liste.
- **L'itération sur les éléments** : les listes permettent d'itérer sur les éléments de la liste.

Ajout d'éléments

Pour ajouter un élément à la fin d'une liste, on utilise la méthode `append()`.

```
liste = [1, 2, 3, 4, 5]
liste.append(6)
```

La liste `liste` contient maintenant les éléments `1`, `2`, `3`, `4`, `5`, et `6`.

On peut également ajouter un élément à un emplacement spécifique dans la liste en utilisant la méthode `insert()`.

```
liste = [1, 2, 3, 4, 5]
liste.insert(2, 10)
```

La liste `liste` contient maintenant les éléments `1`, `10`, `2`, `3`, `4`, et `5`.

Suppression d'éléments

Pour supprimer un élément d'une liste, on utilise la méthode `remove()`.

```
liste = [1, 2, 3, 4, 5]
liste.pop(2)
```

La liste `liste` contient maintenant les éléments `1`, `2`, `4`, et `5`.

Recherche d'éléments

Pour rechercher un élément dans une liste, on utilise la méthode `index()`.

```
liste = [1, 2, 3, 4, 5]
index = liste.index(3)
```

La variable `index` contient la valeur `2`.

Modification d'éléments

Pour modifier un élément dans une liste, on utilise la syntaxe suivante :

```
liste[indice] = nouvelle_valeur
```

```
liste = [1, 2, 3, 4, 5]
liste[2] = 10
```

La liste `liste` contient maintenant les éléments `1`, `2`, `10`, `4`, et `5`.

Itération sur les éléments

Pour itérer sur les éléments d'une liste, on utilise une boucle `for`.

```
liste = [1, 2, 3, 4, 5]
for element in liste:
    print(element)
```

Ce code imprime les éléments de la liste `liste` un par un.

Tuples

Les tuples sont des collections ordonnées qui sont immuables.

Les tuples sont définis en utilisant le symbole `()`. Les éléments d'un tuple sont séparés par des virgules.

Exemple

Voici un exemple de tuple :

```
tuple = (1, 2, 3, 4, 5)
```

Le tuple `tuple` contient les éléments `1`, `2`, `3`, `4`, et `5`.

Opérations sur les tuples

Les tuples supportent un ensemble limité d'opérations, dont :

- **L'accès aux éléments** : les tuples permettent d'accéder aux éléments de manière indexée.
- **La comparaison** : les tuples peuvent être comparés entre eux.
- **La conversion en liste** : les tuples peuvent être convertis en listes.

Accès aux éléments

Pour accéder à un élément d'un tuple, on utilise la syntaxe suivante :

```
tuple[indice]
```

```
tuple = (1, 2, 3, 4, 5)
element = tuple[2]
```

La variable `element` contient la valeur `3`.

Comparaison

Les tuples peuvent être comparés entre eux de manière lexicographique.

```
tuple1 = (1, 2, 3)
tuple2 = (1, 2, 4)

print(tuple1 == tuple2)
```

Ce code imprime la valeur `False`.

Conversion en liste

Les tuples peuvent être convertis en listes en utilisant la fonction `list()`.

```
tuple = (1, 2, 3)
liste = list(tuple)
```

La variable `liste` contient la liste `[1, 2, 3]`.

Les dictionnaires

Les dictionnaires sont des collections associatives qui associent des clés à des valeurs.

Les dictionnaires sont définis en utilisant le symbole `{}`. Les clés sont séparées des valeurs par des deux-points (`:`).

Exemple

Voici un exemple de dictionnaire :

```
dictionnaire = {"nom": "John Doe", "age": 30}
```

Le dictionnaire `dictionnaire` contient la clé `nom` associée à la valeur `John Doe` et la clé `age` associée à la valeur `30`.

Opérations sur les dictionnaires

Les dictionnaires supportent un large éventail d'opérations, dont :

- **L'accès aux éléments** : les dictionnaires permettent d'accéder aux éléments de manière indexée par clé.
- **L'ajout d'éléments** : les dictionnaires permettent d'ajouter des éléments.
- **La suppression d'éléments** : les dictionnaires permettent de supprimer des éléments.
- **La recherche d'éléments** : les dictionnaires permettent de rechercher des éléments.
- **La modification d'éléments** : les dictionnaires permettent de modifier des éléments.
- **L'itération sur les éléments** : les dictionnaires permettent d'itérer sur les éléments.

Accès aux éléments

Pour accéder à un élément d'un dictionnaire, on utilise la syntaxe suivante :

```
dictionnaire[clé]
```

```
dictionnaire = {"nom": "John Doe", "age": 30}
nom = dictionnaire["nom"]
```

La variable `nom` contient la valeur `John Doe`.

Ajout d'éléments

Pour ajouter un élément à un dictionnaire, on utilise la syntaxe suivante :

```
dictionnaire[clé] = valeur
```

```
dictionnaire = {"nom": "John Doe", "age": 30}
dictionnaire["ville"] = "Paris"
```

Le dictionnaire `dictionnaire` contient maintenant la clé `ville` associée à la valeur `Paris`.

Suppression d'éléments

Pour supprimer un élément d'un dictionnaire, on utilise la syntaxe suivante :

```
del dictionnaire[clé]
```

```
dictionnaire = {"nom": "John Doe", "age": 30, "ville":
"Paris"}
del dictionnaire["ville"]
```

Le dictionnaire `dictionnaire` contient maintenant les clés `nom` et `age`.

Recherche d'éléments

Pour rechercher un élément dans un dictionnaire, on utilise la méthode `in`.

```
dictionnaire = {"nom": "John Doe", "age": 30}
print("nom" in dictionnaire)
```

Ce code imprime la valeur `True`.

Modification d'éléments

Pour modifier un élément d'un dictionnaire, on utilise la syntaxe suivante :

```
dictionnaire[clé] = nouvelle_valeur
```

```
dictionnaire = {"nom": "John Doe", "age": 30}
dictionnaire["age"] = 40
```

Le dictionnaire `dictionnaire` contient maintenant la clé `age` associée à la valeur `40`.

Itération sur les éléments

Pour itérer sur les éléments d'un dictionnaire, on utilise une boucle `for`.

```
for clé, valeur in dictionnaire.items():
    print(clé, valeur)
```

Ce code imprime les clés et les valeurs du dictionnaire `dictionnaire` un par un.

Les ensembles

Les ensembles sont des collections non ordonnées qui ne contiennent pas d'éléments dupliqués.

Les ensembles sont définis en utilisant le symbole `{}`. Les éléments d'un ensemble sont séparés par des virgules.

Exemple

Voici un exemple d'ensemble :

```
ensemble = {1, 2, 3, 4, 5}
```

L'ensemble `ensemble` contient les éléments `1`, `2`, `3`, `4`, et `5`.

Opérations sur les ensembles

Les ensembles supportent un large éventail d'opérations, dont :

- **L'ajout d'éléments** : les ensembles permettent d'ajouter des éléments.
- **La suppression d'éléments** : les ensembles permettent de supprimer des éléments.
- **La recherche d'éléments** : les ensembles permettent de rechercher des éléments.
- **L'intersection d'ensembles** : les ensembles permettent de trouver l'intersection de deux ensembles.
- **L'union d'ensembles** : les ensembles permettent de trouver l'union de deux ensembles.
- **La différence d'ensembles** : les ensembles permettent de trouver la différence de deux ensembles.
- **La différence symétrique d'ensembles** : les ensembles permettent de trouver la différence symétrique de deux ensembles.

Ajout d'éléments

Pour ajouter un élément à un ensemble, on utilise la méthode `add()`.

```
ensemble.add(6)
```

L'ensemble `ensemble` contient maintenant les éléments `1`, `2`, `3`, `4`, `5`, et `6`.

Suppression d'éléments

Pour supprimer un élément d'un ensemble, on utilise la méthode `remove()`.

```
ensemble.remove(5)
```

L'ensemble `ensemble` contient maintenant les éléments `1`, `2`, `3`, `4`, et `6`.

Recherche d'éléments

Pour rechercher un élément dans un ensemble, on utilise la méthode `in`.

```
ensemble = {1, 2, 3, 4, 5}
print(1 in ensemble)
```

Ce code imprime la valeur `True`.

Intersection d'ensembles

Pour trouver l'intersection de deux ensembles, on utilise la méthode `intersection()`.

```
ensemble1 = {1, 2, 3, 4, 5}
ensemble2 = {2, 3, 4}

intersection = ensemble1.intersection(ensemble2)
```

La variable `intersection` contient l'ensemble `{2, 3, 4}`.

Union d'ensembles

Pour trouver l'union de deux ensembles, on utilise la méthode `union()`.

```
ensemble1 = {1, 2, 3, 4, 5}
ensemble2 = {2, 3, 4}

union = ensemble1.union(ensemble2)
```

La variable `union` contient l'ensemble `{1, 2, 3, 4, 5}`.

Différence d'ensembles

Pour trouver la différence de deux ensembles, on utilise la méthode `difference()`.

```
ensemble1 = {1, 2, 3, 4, 5}
ensemble2 = {2, 3, 4}

difference = ensemble1.difference(ensemble2)
```

La variable `difference` contient l'ensemble `{1, 5}`.

Différence symétrique d'ensembles

Pour trouver la différence symétrique de deux ensembles, on utilise la méthode `symmetric_difference()`.

```
ensemble1 = {1, 2, 3, 4, 5}
ensemble2 = {2, 3, 4}

symmetric_difference =
ensemble1.symmetric_difference(ensemble2)
```

La variable `symmetric_difference` contient l'ensemble `{1, 5}`.

Conclusion

Les ensembles sont une collection puissante qui peut être utilisée pour stocker et manipuler des données de manière efficace. Ils sont particulièrement utiles pour stocker des données uniques, comme les nombres premiers ou les mots d'un dictionnaire.