Semaine 9

Initiation à l'algorithmique et programmation

Revekka Kyriakoglou

Containers

Containers

Un **container** est un nom générique pour définir un objet Python qui contient une collection d'autres objets.

Containers

Containers

Un **container** est un nom générique pour définir un objet Python qui contient une collection d'autres objets.

Exemples : listed, chaînes de caractères, sets, dictionaires . . .

Terminologie

- test d'appartenance
- fonction len()
- itérable
- ordonné
- indexable
- modifiable
- hachable

Terminologie 1/7

- **Test d'appartenance** : Le test d'appartenance vérifie si un élément est présent dans un conteneur. En Python, cela se fait généralement avec l'opérateur `in`.

  Par exemple, `element in conteneur` renvoie `True` si l'élément est trouvé dans le conteneur, sinon `False`.

Terminologie 2/7

- **Fonction len()** : La fonction `len()` est utilisée pour obtenir le nombre d'éléments dans un conteneur.

  Par exemple, `len(conteneur)` renvoie un entier représentant le nombre d'éléments présents dans le conteneur.

Terminologie 3/7

- **Itérable** : Un itérable est un objet sur lequel on peut itérer, c'est-à-dire parcourir ses éléments un par un.
  Les listes, les tuples, les dictionnaires, les ensembles, et les chaînes de caractères sont des exemples d'itérables en Python.

Terminologie 4/7

- **Ordonné** : Un conteneur est dit ordonné si les éléments sont disposés dans un ordre spécifique et que cet ordre est maintenu au fil du temps.

  Par exemple, les listes et les tuples sont ordonnés car les éléments conservent leur position d'insertion.

New

Les dictionnaires sont considérés comme ordonnés à partir de **Python 3.7**.

Terminologie 5/7

- **Indexable** : Un objet est indexable si vous pouvez accéder à ses éléments en utilisant un indice.

  Par exemple, les listes et les tuples sont d'objets indexables, où vous pouvez utiliser un indice pour accéder à un élément spécifique (par exemple, `conteneur[indice]`).

Terminologie 6/7

- **Modifiable** : Un objet est dit non **modifiable** lorsqu'on ne peut pas le modifier, ou lorsqu'on ne peut pas en modifier un de ses éléments si c'est un container. On parle aussi d'objet **immuable** (immutable object en anglais). Cela signifie qu'une fois créé, Python ne permet plus de le modifier par la suite.

  Les listes et les dictionnaires sont modifiables car vous pouvez ajouter, supprimer ou modifier leurs éléments.

Terminologie 7/7

- **Hachable** : Dire qu'un objet Python est **hachable** signifie qu'on peut lui donner une sorte d'empreinte digitale unique, grâce à une fonction spéciale appelée `hash()`. Cette valeur de hachage, est un nombre qui représente de manière unique le contenu de l'objet. Si deux objets sont exactement les mêmes, ils auront la même empreinte digitale.

Contrairement à l'identifiant d'un objet, qui est comme son numéro de carte d'identité (`id()`) et qui est donné par Python quand l'objet est créé, la valeur de hachage est calculée à partir de ce que contient l'objet. Tous les objets ont un identifiant, mais seuls certains objets, ceux qui sont hachables, peuvent avoir une valeur de hachage.

La valeur de hachage est utile, par exemple, quand on veut utiliser cet objet comme clé dans un dictionnaire, car cela assure que chaque clé est unique.

```
hash("PYTHON")

hash(3)

hash(3.1415)

hash([1, 2, 3])
```

Liste

- **Test d'appartenance** : Oui, on peut vérifier si un élément est présent dans une liste avec l'opérateur `in`.

Liste

- **Test d'appartenance** : Oui, on peut vérifier si un élément est présent dans une liste avec l'opérateur `in`.
- **Fonction len()** : Oui, on peut utiliser `len(liste)` pour obtenir le nombre d'éléments dans la liste.

Liste

- **Test d'appartenance** : Oui, on peut vérifier si un élément est présent dans une liste avec l'opérateur `in`.
- **Fonction len()** : Oui, on peut utiliser `len(liste)` pour obtenir le nombre d'éléments dans la liste.
- **Itérable** : Oui, on peut itérer sur les éléments d'une liste avec une boucle, par exemple `for element in liste`.

Liste

- **Test d'appartenance** : Oui, on peut vérifier si un élément est présent dans une liste avec l'opérateur `in`.
- **Fonction len()** : Oui, on peut utiliser `len(liste)` pour obtenir le nombre d'éléments dans la liste.
- **Itérable** : Oui, on peut itérer sur les éléments d'une liste avec une boucle, par exemple `for element in liste`.
- **Ordonné** : Oui, les éléments dans une liste sont maintenus dans l'ordre où ils ont été ajoutés.

Liste

- **Test d'appartenance** : Oui, on peut vérifier si un élément est présent dans une liste avec l'opérateur `in`.
- **Fonction len()** : Oui, on peut utiliser `len(liste)` pour obtenir le nombre d'éléments dans la liste.
- **Itérable** : Oui, on peut itérer sur les éléments d'une liste avec une boucle, par exemple `for element in liste`.
- **Ordonné** : Oui, les éléments dans une liste sont maintenus dans l'ordre où ils ont été ajoutés.
- **Indexable** : Oui, on peut accéder à un élément par son index, par exemple `liste[0]` pour le premier élément.

Liste

- **Test d'appartenance** : Oui, on peut vérifier si un élément est présent dans une liste avec l'opérateur `in`.
- **Fonction len()** : Oui, on peut utiliser `len(liste)` pour obtenir le nombre d'éléments dans la liste.
- **Itérable** : Oui, on peut itérer sur les éléments d'une liste avec une boucle, par exemple `for element in liste`.
- **Ordonné** : Oui, les éléments dans une liste sont maintenus dans l'ordre où ils ont été ajoutés.
- **Indexable** : Oui, on peut accéder à un élément par son index, par exemple `liste[0]` pour le premier élément.
- **Modifiable** : Oui, on peut modifier les éléments d'une liste, ajouter ou supprimer des éléments après sa création.

Liste

- **Test d'appartenance** : Oui, on peut vérifier si un élément est présent dans une liste avec l'opérateur `in`.
- **Fonction len()** : Oui, on peut utiliser `len(liste)` pour obtenir le nombre d'éléments dans la liste.
- **Itérable** : Oui, on peut itérer sur les éléments d'une liste avec une boucle, par exemple `for element in liste`.
- **Ordonné** : Oui, les éléments dans une liste sont maintenus dans l'ordre où ils ont été ajoutés.
- **Indexable** : Oui, on peut accéder à un élément par son index, par exemple `liste[0]` pour le premier élément.
- **Modifiable** : Oui, on peut modifier les éléments d'une liste, ajouter ou supprimer des éléments après sa création.
- **Hachable** : Non, une liste ne peut pas être utilisée comme clé de dictionnaire car elle est modifiable, et donc sa valeur de hachage pourrait changer.

Chaîne de Caractères 1/2

- **Test d'appartenance** : Oui, on peut vérifier si un caractère ou une sous-chaîne est présent dans une chaîne avec l'opérateur `in`.

Chaîne de Caractères 1/2

- **Test d'appartenance** : Oui, on peut vérifier si un caractère ou une sous-chaîne est présent dans une chaîne avec l'opérateur `in`.
- **Fonction len()** : Oui, on peut utiliser `len(chaine)` pour obtenir le nombre de caractères dans la chaîne.

Chaîne de Caractères 1/2

- **Test d'appartenance** : Oui, on peut vérifier si un caractère ou une sous-chaîne est présent dans une chaîne avec l'opérateur `in`.
- **Fonction len()** : Oui, on peut utiliser `len(chaine)` pour obtenir le nombre de caractères dans la chaîne.
- **Itérable** : Oui, on peut itérer sur chaque caractère d'une chaîne avec une boucle, par exemple `for caractere in chaine`.

Chaîne de Caractères 1/2

- **Test d'appartenance** : Oui, on peut vérifier si un caractère ou une sous-chaîne est présent dans une chaîne avec l'opérateur `in`.
- **Fonction len()** : Oui, on peut utiliser `len(chaine)` pour obtenir le nombre de caractères dans la chaîne.
- **Itérable** : Oui, on peut itérer sur chaque caractère d'une chaîne avec une boucle, par exemple `for caractere in chaine`.
- **Ordonné** : Oui, les caractères dans une chaîne sont maintenus dans l'ordre spécifié lors de sa création.

Chaîne de Caractères 2/2

- **Indexable** : Oui, on peut accéder à un caractère par son index, par exemple `chaine[0]` pour le premier caractère.

Chaîne de Caractères 2/2

- **Indexable** : Oui, on peut accéder à un caractère par son index, par exemple `chaine[0]` pour le premier caractère.
- **Modifiable** : Non, une fois créée, la valeur d'une chaîne de caractères ne peut pas être modifiée (les chaînes sont immuables). Pour modifier une chaîne, il faut créer une nouvelle chaîne.

Chaîne de Caractères 2/2

- **Indexable** : Oui, on peut accéder à un caractère par son index, par exemple `chaine[0]` pour le premier caractère.
- **Modifiable** : Non, une fois créée, la valeur d'une chaîne de caractères ne peut pas être modifiée (les chaînes sont immuables). Pour modifier une chaîne, il faut créer une nouvelle chaîne.
- **Hachable** : Oui, puisque les chaînes de caractères sont immuables, elles peuvent être utilisées comme clés de dictionnaire.

Définition des Dictionnaires

Dictionnaires

Un dictionnaire en Python est une collection non ordonnée (bien que l'ordre d'insertion soit préservé depuis Python 3.7), modifiable et indexée. Dans un dictionnaire, les données sont stockées sous forme de paires clé-valeur. Chaque clé est unique et associée à une valeur.

Définition des Dictionnaires

Dictionnaires

Un dictionnaire en Python est une collection non ordonnée (bien que l'ordre d'insertion soit préservé depuis Python 3.7), modifiable et indexée. Dans un dictionnaire, les données sont stockées sous forme de paires clé-valeur. Chaque clé est unique et associée à une valeur.

Les dictionnaires sont particulièrement utiles pour stocker et accéder à des données de manière efficace lorsque ces données sont associées à des identifiants uniques ou des clés.

Syntaxe des Dictionnaires en Python 1/2

Création d'un Dictionnaire

- Dictionnaire vide : `mon_dictionnaire = {}`
- Dictionnaire avec des paires clé-valeur : `mon_dictionnaire = {'cle1': 'valeur1', 'cle2': 'valeur2'}`

Accès aux Valeurs

- Accéder à une valeur : `valeur = mon_dictionnaire['cle1']`

Syntaxe des Dictionnaires en Python 2/2

Ajout ou Modification de Paires Clé-Valeur

- Ajouter ou modifier : `mon_dictionnaire['cle3'] = 'valeur3'`

Suppression de Paires Clé-Valeur

- Supprimer une paire clé-valeur : `del mon_dictionnaire['cle3']`

Cette structure clé-valeur rend les dictionnaires idéaux pour stocker des données de manière organisée et permettre un accès rapide à la valeur associée à chaque clé.

Propriétés des Dictionnaires

- **Accès aux éléments** : On accède aux éléments en utilisant leurs clés, par exemple `dictionnaire[cle]`.
- **Modifiable** : Il est possible d'ajouter, de modifier ou de supprimer des paires clé-valeur après la création du dictionnaire.
- **Itérable** : On peut itérer sur les clés, les valeurs ou les paires clé-valeur d'un dictionnaire.
- **Clés uniques** : Chaque clé dans un dictionnaire doit être unique.
- **Clés hachables** : Les clés d'un dictionnaire doivent être de types immuables (comme les chaînes de caractères, les nombres ou les tuples contenant uniquement des éléments immuables) pour garantir leur unicité et immuabilité.

Exemple sur les Dictionnaires

Considérons un dictionnaire stockant les informations de contact d'amis, où les noms des amis servent de clés et leurs numéros de téléphone comme valeurs.

```
# Creation d'un dictionnaire
contacts = {}

# Ajout de quelques contacts dans le dictionnaire
contacts['Alice'] = '123-456-7890'
contacts['Bob'] = '987-654-3210'
contacts['Charlie'] = '555-666-7777'

# Afficher le dictionnaire de contacts
print(contacts)
```

Méthodes : get et del

La méthode get()

Permet d'accéder à la valeur d'une clé donnée de manière sûre, en retournant une valeur par défaut si la clé n'existe pas.

```
valeur = mon_dictionnaire.get('cle', 'valeur par défaut')
```


Cela évite une exception si la clé n'est pas trouvée.

L'instruction del

Permet de supprimer une paire clé-valeur du dictionnaire.

```
del mon_dictionnaire['cle']
```


Si la clé n'existe pas, une exception sera levée, donc soyez prudent lors de son utilisation.

Exercice : Gestion d'un Inventaire de Produits

Instructions

1. **Création de l'Inventaire :** Créez un dictionnaire nommé `inventaire` avec des produits comme clés et leurs quantités en stock comme valeurs.
2. **Affichage de la Quantité :** Demandez le nom d'un produit et affichez sa quantité. Si le produit n'est pas trouvé, indiquez qu'il est indisponible.
3. **Mise à Jour de l'Inventaire :** Mettez à jour la quantité d'un produit existant ou ajoutez un nouveau produit avec sa quantité sur saisie de l'utilisateur.
4. **Suppression d'un Produit :** Supprimez un produit de l'inventaire sur demande de l'utilisateur.
5. **Affichage de l'Inventaire :** Montrez l'inventaire complet après les mises à jour et suppressions.

Conseils :
Utilisez `input()` pour la saisie, `print()` pour l'affichage, et gérez les cas où un produit n'existe pas avec soin.

```
# Creation de dictionnaire
inventaire = {'pommes': 30, 'bananes': 15, 'oranges': 22}
# Affichage de la quantite d'un produit
produit = input("Entrez le nom d'un produit: ")
if produit in inventaire:
    print(f"Quantite de {produit}: {inventaire[produit]}")
else:
    print(f"Le produit {produit} est indisponible.")
# Mise a jour de l'inventaire
produit = input("Entrez le produit a mettre a jour: ")
quantite = int(input("Entrez la quantite: "))
inventaire[produit] = inventaire.get(produit, 0) + quantite
# Suppression d'un produit
produit = input("Entrez le nom du produit a supprimer: ")
if produit in inventaire:
    del inventaire[produit]
else:
    print(f"Le produit {produit} n'existe pas.")
# Affichage de l'inventaire complet
for produit, quantite in inventaire.items():
    print(f"{produit}: {quantite}")
```

Méthodes : .keys(), .values() et .items()

.keys()

Retourne une vue des clés du dictionnaire.

```
for cle in mon_dictionnaire.keys():
    print(cle)
```

.values()

Retourne une vue des valeurs du dictionnaire.

```
for valeur in mon_dictionnaire.values():
    print(valeur)
```

Méthodes : .keys(), .values() et .items()

.items()

Retourne une vue des paires clé-valeur sous forme de tuples.

```
for cle, valeur in mon_dictionnaire.items():
    print(f"{cle}: {valeur}")
```

max() et min()

max() et min()

Les fonctions `min()` et `max()` acceptent l'argument `key=`. On peut ainsi obtenir la clé associée au minimum ou au maximum des valeurs d'un dictionnaire.

```
dico = {"a": 15, "b": 5, "c":20}
print(max(dico, key=dico.get))
print(min(dico, key=dico.get))
```

Travailler avec une Liste de Dictionnaires en Python

Une liste de dictionnaires est un moyen puissant de stocker et de manipuler une collection d'ensembles de données clé-valeur en Python. Chaque élément de la liste est un dictionnaire représentant un objet ou un enregistrement distinct.

Exemple de Liste de Dictionnaires

```
employes = [
    {'nom': 'Alice', 'poste': 'Ingénieur', 'age': 30},
    {'nom': 'Bob', 'poste': 'Designer', 'age': 25},
    {'nom': 'Charlie', 'poste': 'Manager', 'age': 35}
]
```

Accès et Modification

Accéder à un dictionnaire spécifique :

```
# Affiche les informations d'Alice
print(employes[0])
```

Modifier une valeur :

```
# Change l'âge de Bob
employes[1]['age'] = 26
```

Ajout et Suppression d'Éléments

Ajouter un nouvel employé :

```
employes.append({'nom': 'Diana', 'poste': 'RH', 'age': 28})
```

Supprimer un employé :

```
# Supprime Charlie de la liste
del employes[2]
```

Cette structure est particulièrement utile pour représenter des données complexes de manière organisée et facilement accessible.