Semaine 7

Initiation à l'algorithmique et programmation

Revekka Kyriakoglou

Plan du cours

- Listes

Définition


Une liste est une structure de données qui contient une série de valeurs (peut-être de types différents).


Une liste est déclarée par une série de valeurs séparées par des virgules, et le tout encadré par des crochets.

Motivation

- **Regrouper les Données :** Les listes vous permettent de regrouper des informations liées. Par exemple, vous pouvez utiliser une liste pour représenter une collection d'étudiants, une séquence de nombres ou une série de tâches.

Motivation

- **Regrouper les Données :** Les listes vous permettent de regrouper des informations liées. Par exemple, vous pouvez utiliser une liste pour représenter une collection d'étudiants, une séquence de nombres ou une série de tâches.

```
etudiants = ["Alice", "Bob", "Charlie"]
nombres = [1, 2, 3, 4, 5]
```

Motivation

- **Indexation et Accès aux Éléments :** Les listes sont ordonnées, ce qui signifie que chaque élément a une position ou un indice spécifique. Vous pouvez accéder aux éléments individuels par leur indice, en commençant par 0.

Motivation

- **Indexation et Accès aux Éléments :** Les listes sont ordonnées, ce qui signifie que chaque élément a une position ou un indice spécifique. Vous pouvez accéder aux éléments individuels par leur indice, en commençant par 0.

```
etudiants = ["Alice", "Bob", "Charlie"]
nombres = [1, 2, 3, 4, 5]

print(etudiants[0])
print(nombres[2])
```

Motivation

- **Taille Dynamique :** Contrairement à certains autres langages de programmation, les listes Python peuvent grandir ou rétrécir dynamiquement. Vous pouvez ajouter ou supprimer des éléments facilement.

Motivation

- **Taille Dynamique :** Contrairement à certains autres langages de programmation, les listes Python peuvent grandir ou rétrécir dynamiquement. Vous pouvez ajouter ou supprimer des éléments facilement.

```
etudiants = ["NDIAYE", "LEVY"]

# Ajouter un nouvel  tudiant
etudiants.append("ANGOT")
# Supprimer une t che
taches.remove("NDIAYE")
```

Motivation

- **Itération :** Les listes sont itérables, ce qui facilite le parcours de chaque élément. C'est particulièrement utile lors de l'exécution de tâches répétitives ou d'opérations sur chaque élément de la liste.

Motivation

- **Itération :** Les listes sont itérables, ce qui facilite le parcours de chaque élément. C'est particulièrement utile lors de l'exécution de tâches répétitives ou d'opérations sur chaque élément de la liste.

```
etudiants = ["HUGO", "MOLIERE", "MODIANO"]
for etudiant in etudiants:
    print(f"Bonjour␣{etudiant}␣!")
```

Motivation

- **Polyvalence :** Les listes peuvent contenir un mélange de types de données, vous permettant de créer des structures complexes et des représentations variées de l'information.

Motivation

- **Polyvalence :** Les listes peuvent contenir un mélange de types de données, vous permettant de créer des structures complexes et des représentations variées de l'information.

```
liste_mixte = [14531, "HUGO", True, [12, 13, 14, 20]]
```

Motivation

- **Manipulation des Données :** Les listes offrent de nombreuses méthodes pour manipuler les données qu'elles contiennent. Vous pouvez les trier, les inverser, voire même les filtrer facilement.

Motivation

- **Manipulation des Données :** Les listes offrent de nombreuses méthodes pour manipuler les données qu'elles contiennent. Vous pouvez les trier, les inverser, voire même les filtrer facilement.

```
# Create a new list with even numbers
nombres_pairs = [x for x in nombres if x % 2 == 0]
```

Opération sur les listes

Tout comme les chaînes de caractères, les listes supportent l'opérateur $+$ de concaténation, ainsi que l'opérateur $*$ pour la duplication.

Opération sur les listes

Tout comme les chaînes de caractères, les listes supportent l'opérateur $+$ de concaténation, ainsi que l'opérateur $*$ pour la duplication.

Que fait le code suivant ?

```
nombres_pairs = [x for x in nombres if x % 2 == 0]
nombres_impairs = [x for x in nombres if x % 2 != 0]
l_concat = nombres_pairs + nombres_impairs
print(l_concat)
l_dupl = nombres_pairs*2
print(l_dupl)
```

Tranche de liste

Tout comme les chaînes de caractères, il est possible de sélectionner une partie d'une liste en utilisant un indiçage construit sur le modèle `liste[m:n+1]` pour récupérer tous les éléments de position m jusquà n (inclu).

Tranche de liste

Tout comme les chaînes de caractères, il est possible de sélectionner une partie d'une liste en utilisant un indiçage construit sur le modèle `liste[m:n+1]` pour récupérer tous les éléments de position m jusquà n (inclu).

? Que fait le code suivant ?

```
nombres_pairs = [x for x in nombres if x % 2 == 0]
l = liste[::-1]
print(l)
```

Tranche de liste

Tout comme les chaînes de caractères, il est possible de sélectionner une partie d'une liste en utilisant un indiçage construit sur le modèle `liste[m:n+1]` pour récupérer tous les éléments de position m jusquà n (inclu).


Que fait le code suivant ?

```
nombres_pairs = [x for x in nombres if x % 2 == 0]
l = liste[::-1]
print(l)
```


`len()` vous permet de connaître la longueur d'une liste.

Exercice 1 : Liste de Nombres Pairs

Écrivez une fonction appelée `nombres_pairs` qui prend un nombre entier `n` en tant que paramètre et renvoie une liste contenant les nombres pairs de 0 à *n* inclus.

Exercice 1 : Liste de Nombres Pairs

Écrivez une fonction appelée `nombres_pairs` qui prend un nombre entier `n` en tant que paramètre et renvoie une liste contenant les nombres pairs de 0 à *n* inclus.

SOLUTION :

```
def nombres_pairs(n):
    liste_pairs = []
    i = 0
    while i <= n:
        liste_pairs.append(i)
        i = i +2
    return liste_pairs

def nombres_pairs_2(n):
    return [i for i in range(n + 1) if i % 2 == 0]
```

Exercice 2 : Concaténation de Listes

Écrivez une fonction appelée `concatener_listes` qui prend deux listes en tant que paramètres et renvoie une nouvelle liste qui est la concaténation des deux.

Exercice 2 : Concaténation de Listes

Écrivez une fonction appelée `concatener_listes` qui prend deux listes en tant que paramètres et renvoie une nouvelle liste qui est la concaténation des deux.

SOLUTION :

```
def concatener_listes(l1, l2):
    liste_concat = l1 +l2
    return liste_concat
```

Exercice 3 : Suppression d'Éléments Duplicates

Écrivez une fonction appelée supprimer_duplicates qui prend une liste en tant que paramètre et renvoie une nouvelle liste ne contenant que les éléments uniques, dans l'ordre d'apparition.

Exercice 3 : Suppression d'Éléments Duplicates

Écrivez une fonction appelée supprimer_duplicates qui prend une liste en tant que paramètre et renvoie une nouvelle liste ne contenant que les éléments uniques, dans l'ordre d'apparition.

SOLUTION :

```
def supprimer_duplicates(l):
    l_sans_duplicates = []
    for elem in l:
        if elem not in l_sans_duplicates:
            l_sans_duplicates += [elem]
    return l_sans_duplicates
```

Méthodes

Les listes possèdent de nombreuses méthodes ! ! !


On rappelle qu'une méthode est une fonction qui agit sur l'objet auquel elle est attachée par un point.

Méthodes

Les listes possèdent de nombreuses méthodes !!!


On rappelle qu'une méthode est une fonction qui agit sur l'objet auquel elle est attachée par un point.

- `.append()` : ajoute un élément à la fin d'une liste.
- `.insert()` : insère un objet dans une liste à un indice déterminé.
- `del` supprime un élément d'une liste à un indice déterminé.
- `.remove()` : supprime un élément d'une liste à partir de sa valeur.
- `.sort()` trie les éléments d'une liste du plus petit au plus grand.
- `sorted()` : trie également une liste.
  Contrairement à la méthode précédente .sort(), cette fonction renvoie la liste triée et ne modifie pas la liste initiale.
- `.reverse()` : inverse une liste.
- `.count()` : compte le nombre d'éléments (passés en argument) dans une liste.

Listes de listes

Les listes de listes sont une structure de données utilisée pour représenter des tableaux bidimensionnels ou des matrices. Chaque élément d'une liste de listes est lui-même une liste.

Listes de listes

Les listes de listes sont une structure de données utilisée pour représenter des tableaux bidimensionnels ou des matrices. Chaque élément d'une liste de listes est lui-même une liste.

```
#creation
et1 = ["Hugo", 14]
et2 = ["Molier", 12]
et3 = ["Levy", 15]
notes = [et1, et2, et3]
# same as
# notes = [["Hugo", 14], ["Molier", 12], ["Levy", 15]]

print(notes[1])
print(notes[1][0])
```

Exercice 4 : Création de Matrice

Écrivez une fonction `creer_matrice` qui prend deux entiers `m` et `n` en tant que paramètres et renvoie une matrice (`m` lignes et `n` colonnes) remplie de zéros.

Exercice 4 : Création de Matrice

Écrivez une fonction `creer_matrice` qui prend deux entiers `m` et `n` en tant que paramètres et renvoie une matrice (`m` lignes et `n` colonnes) remplie de zéros.

SOLUTION :

```
def creer_matrice(m, n):
    matrice = []
    for i in range(m):
        matrice.append([0]*n)
    return matrice
```

Exercice 5 : Somme des Éléments

Écrivez une fonction `somme_elements` qui prend une matrice en tant que paramètre et renvoie la somme de tous les éléments de la matrice.

Exercice 5 : Somme des Éléments

Écrivez une fonction `somme_elements` qui prend une matrice en tant que paramètre et renvoie la somme de tous les éléments de la matrice.

SOLUTION :

```
def somme_elements(matrice):
    somme = 0
    for i in range(len(matrice)):
        for j in range(len(matrice[i])):
            somme += matrice[i][j]
    return somme
```

Exercice 6 : Multiplication par un Scalaire

Écrivez une fonction `multiplier_par_scalaire` qui prend une matrice et un scalaire en tant que paramètres, et renvoie une nouvelle matrice résultant de la multiplication de chaque élément par le scalaire.

Exemple :

```
>>> matrice = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> multiplier_par_scalaire(matrice, 2)
[[2, 4, 6], [8, 10, 12], [14, 16, 18]]
```