Atelier d'informatique

Épisode II : Structures de données

30 janvier 2017

Introduction

Dans ce chapitre, nous allons voir la notion de *structure de données*, permettant de stocker et organiser un ensemble de données au sein d'un seul objet dans la mémoire de l'ordinateur, avec un type spécifique permettant de les traiter.

Introduction

Dans ce chapitre, nous allons voir la notion de *structure de données*, permettant de stocker et organiser un ensemble de données au sein d'un seul objet dans la mémoire de l'ordinateur, avec un type spécifique permettant de les traiter.
On verra principalement le type `list` implémenté par défaut dans Python.

Listes Python

Bases

Sous Python, une liste `l` une collection d'éléments de la forme $e = (\texttt{mem}, \texttt{suivant})$ où `mem` est l'adresse mémoire d'une valeur, et `suivant` un pointeur vers le successeur de e au sein de la liste `l`. Ainsi chaque élément contient où trouver la valeur correspondante et où trouver l'élément suivant.

Listes Python

Bases

Sous Python, une liste `l` une collection d'éléments de la forme $e = (\texttt{mem}, \texttt{suivant})$ où `mem` est l'adresse mémoire d'une valeur, et `suivant` un pointeur vers le successeur de e au sein de la liste `l`. Ainsi chaque élément contient où trouver la valeur correspondante et où trouver l'élément suivant.
Pour définir une liste en Python, plusieurs méthodes son possibles. On va commencer avec la définition d'une liste par explicitation : on énumère les éléments de la liste que l'on veut, entre des crochets `[` et `]`, et séparés par une virgule (`,`).

Listes Python

Bases

Sous Python, une liste `l` une collection d'éléments de la forme $e = (\texttt{mem}, \texttt{suivant})$ où `mem` est l'adresse mémoire d'une valeur, et `suivant` un pointeur vers le successeur de e au sein de la liste `l`. Ainsi chaque élément contient où trouver la valeur correspondante et où trouver l'élément suivant.
Pour définir une liste en Python, plusieurs méthodes son possibles. On va commencer avec la définition d'une liste par explicitation : on énumère les éléments de la liste que l'on veut, entre des crochets `[` et `]`, et séparés par une virgule (`,`).
Il n'est pas obligé d'entrer des valeurs du même type, même si cela n'est pas vraiment conseillé.

Listes Python

Bases

Sous Python, une liste `l` une collection d'éléments de la forme $e = (\texttt{mem}, \texttt{suivant})$ où `mem` est l'adresse mémoire d'une valeur, et `suivant` un pointeur vers le successeur de e au sein de la liste `l`. Ainsi chaque élément contient où trouver la valeur correspondante et où trouver l'élément suivant.
Pour définir une liste en Python, plusieurs méthodes son possibles. On va commencer avec la définition d'une liste par explicitation : on énumère les éléments de la liste que l'on veut, entre des crochets `[` et `]`, et séparés par une virgule (`,`).
Il n'est pas obligé d'entrer des valeurs du même type, même si cela n'est pas vraiment conseillé.

Exemple

Dans la console, entrez l'instruction `l = [0,1,2,3,4,5]`. Affichez `l`.

Listes Python

Bases

Sous Python, une liste `l` une collection d'éléments de la forme $e = (\texttt{mem}, \texttt{suivant})$ où `mem` est l'adresse mémoire d'une valeur, et `suivant` un pointeur vers le successeur de e au sein de la liste `l`. Ainsi chaque élément contient où trouver la valeur correspondante et où trouver l'élément suivant.
Pour définir une liste en Python, plusieurs méthodes son possibles. On va commencer avec la définition d'une liste par explicitation : on énumère les éléments de la liste que l'on veut, entre des crochets `[` et `]`, et séparés par une virgule (`,`).
Il n'est pas obligé d'entrer des valeurs du même type, même si cela n'est pas vraiment conseillé.

Exemple

Dans la console, entrez l'instruction `l = [0,1,2,3,4,5]`. Affichez `l`. Évaluez l'expression `len(l)`. Que remarquez-vous ?

La fonction `len` permet en effet d'évaluer la longueur d'une liste, comme elle servait à évaluer celle d'une chaîne de caractères.

Listes Python

Accéder à un élément dans une liste

Pour accéder à un élément d'une liste, on utilise la syntaxe `l[i]`, où `i` est la position de l'élément de la liste qui nous intéresse... mais attention, on commence à compter à partir de zéro !

Listes Python

Accéder à un élément dans une liste

Pour accéder à un élément d'une liste, on utilise la syntaxe `l[i]`, où `i` est la position de l'élément de la liste qui nous intéresse... mais attention, on commence à compter à partir de zéro !

Exemple

Que vaut `l[0]` ? Tentez d'évaluer `l[6]`.

Listes Python

Accéder à un élément dans une liste

Pour accéder à un élément d'une liste, on utilise la syntaxe `l[i]`, où `i` est la position de l'élément de la liste qui nous intéresse... mais attention, on commence à compter à partir de zéro !

Exemple

Que vaut `l[0]` ? Tentez d'évaluer `l[6]`.

On peut même compter à rebours...

Exemple

Que vaut `l[-1]` ? Et `l[-2]` ?

Listes Python

Accéder à un élément dans une liste

Pour accéder à un élément d'une liste, on utilise la syntaxe `l[i]`, où `i` est la position de l'élément de la liste qui nous intéresse... mais attention, on commence à compter à partir de zéro !

Exemple

Que vaut `l[0]` ? Tentez d'évaluer `l[6]`.

On peut même compter à rebours...

Exemple

Que vaut `l[-1]` ? Et `l[-2]` ?

Pour obtenir une « sous-liste », ou un *slicing*, des termes de `i` à `j-1`, on utilise la syntaxe `l[i :j]`.

Listes Python

Accéder à un élément dans une liste

Pour accéder à un élément d'une liste, on utilise la syntaxe `l[i]`, où `i` est la position de l'élément de la liste qui nous intéresse... mais attention, on commence à compter à partir de zéro !

Exemple

Que vaut `l[0]` ? Tentez d'évaluer `l[6]`.

On peut même compter à rebours...

Exemple

Que vaut `l[-1]` ? Et `l[-2]` ?

Pour obtenir une « sous-liste », ou un *slicing*, des termes de `i` à `j-1`, on utilise la syntaxe `l[i :j]`. Pour ne prendre qu'un élément sur 2, on utilise `l[i :j :2]` ou plus généralement `l[i :j :k]` pour n'en prendre que un sur `k`.

Listes Python

Accéder à un élément dans une liste

Pour accéder à un élément d'une liste, on utilise la syntaxe `l[i]`, où `i` est la position de l'élément de la liste qui nous intéresse... mais attention, on commence à compter à partir de zéro !

Exemple

Que vaut `l[0]` ? Tentez d'évaluer `l[6]`.

On peut même compter à rebours...

Exemple

Que vaut `l[-1]` ? Et `l[-2]` ?

Pour obtenir une « sous-liste », ou un *slicing*, des termes de `i` à `j-1`, on utilise la syntaxe `l[i :j]`. Pour ne prendre qu'un élément sur 2, on utilise `l[i :j :2]` ou plus généralement `l[i :j :k]` pour n'en prendre que un sur `k`.
Les valeurs par défaut (utilisées quand rien n'est précisé) de `i`, `j` et `k` sont respectivement 0, l'indice de fin de la liste, et 1.

Listes Python

Modifier une liste

Les listes Python sont des objets dits *mutables* : on peut modifier les valeurs qu'ils contiennent.

Listes Python

Modifier une liste

Les listes Python sont des objets dits *mutables* : on peut modifier les valeurs qu'ils contiennent.
En Python, on peut modifier un élément d'une liste en réaffectant la valeur comme on le ferait avec une variable.

Exemple

Entrez `l[0] = "banane"` et affichez `l`. Que remarquez-vous ?

Listes Python

Modifier une liste

Les listes Python sont des objets dits *mutables* : on peut modifier les valeurs qu'ils contiennent.
En Python, on peut modifier un élément d'une liste en réaffectant la valeur comme on le ferait avec une variable.

Exemple

Entrez `l[0] = "banane"` et affichez `l`. Que remarquez-vous ?

Mais, une liste n'est qu'une collection d'adresses mémoire et pointeurs !

Exemple

Copiez la liste `l` dans une autre variable `t`. Exécutez l'instruction `t[0] = "Tartiflette"`. Affichez `t` pour vérifier que l'affectation a été faite.

Listes Python

Modifier une liste

Les listes Python sont des objets dits *mutables* : on peut modifier les valeurs qu'ils contiennent.
En Python, on peut modifier un élément d'une liste en réaffectant la valeur comme on le ferait avec une variable.

Exemple

Entrez `l[0] = "banane"` et affichez `l`. Que remarquez-vous ?

Mais, une liste n'est qu'une collection d'adresses mémoire et pointeurs !

Exemple

Copiez la liste `l` dans une autre variable `t`. Exécutez l'instruction `t[0] = "Tartiflette"`. Affichez `t` pour vérifier que l'affectation a été faite.
Maintenant, affichez `l`. Que remarquez-vous ?

Il existe une parade, en faisant un *slicing* : `t = l[ :]`. Mais ce n'est pas tout à fait idéal...

Listes Python

Modifier une liste (suite)

Enfin, on peut ajouter une valeur à une liste en utilisant la méthode `append`, qui s'utilise comme suit :

```
l.append(x)
```

ajoute la valeur `x` à la fin de la liste `l`.

Exemple

Effectuez l'instruction `l.append(18)`, et affichez `l`. Vérifiez la longueur via `len`.

Boucles
Introduction

Des fois, on veut exécuter plusieurs fois les mêmes instructions au sein d'un programme, et le nombre de fois peut soit être très grand, soit dépendre des variables en jeu.

Boucles
Introduction

Des fois, on veut exécuter plusieurs fois les mêmes instructions au sein d'un programme, et le nombre de fois peut soit être très grand, soit dépendre des variables en jeu.
En tout cas, on ne va pas réécrire plusieurs fois ces mêmes instructions.
Pour cela, on utilise des boucles.

Boucles
Introduction

Des fois, on veut exécuter plusieurs fois les mêmes instructions au sein d'un programme, et le nombre de fois peut soit être très grand, soit dépendre des variables en jeu.
En tout cas, on ne va pas réécrire plusieurs fois ces mêmes instructions.
Pour cela, on utilise des boucles.
Python propose deux types de boucle. La boucle itérative, `for`, s'exécutant en lisant les éléments d'une structure de donnée que l'on peut parcourir, telle une liste, et la boucle conditionnelle `while` qui ne s'arrête que lorsqu'une condition booléenne à vérifier prend la valeur `False`.

Boucles
Boucle itérative

Sous Python, elle est implémentée comme suit :

```
for i in itérable :
    < traitement >
```

où `itérable` peut être une liste, par exemple, ou encore une chaîne de caractères, ou généralement tout objet ayant l'attribut `'__iter'`, c'est-à-dire sur lequel on peut itérer.

Boucles
Boucle itérative

Sous Python, elle est implémentée comme suit :

```
for i in itérable :
    < traitement >
```

où `itérable` peut être une liste, par exemple, ou encore une chaîne de caractères, ou généralement tout objet ayant l'attribut `'__iter'`, c'est-à-dire sur lequel on peut itérer.

Exemple

- Écrire un programme qui affiche 20 fois « `J'aime la tartiflette` ». On peut générer un itérable d'entiers allant de 0 à `N-1` via `range(N)`, ou de façon équivalente `range(0,N)`.

Boucles
Boucle itérative

Sous Python, elle est implémentée comme suit :

```
for i in itérable :
    < traitement >
```

où `itérable` peut être une liste, par exemple, ou encore une chaîne de caractères, ou généralement tout objet ayant l'attribut `'__iter'`, c'est-à-dire sur lequel on peut itérer.

Exemple

- Écrire un programme qui affiche 20 fois « `J'aime la tartiflette` ». On peut générer un itérable d'entiers allant de 0 à `N-1` via `range(N)`, ou de façon équivalente `range(0,N)`.
- Écrire un programme qui compte de 1 à 10.

Boucles
Boucle itérative

Sous Python, elle est implémentée comme suit :

```
for i in itérable :
    < traitement >
```

où `itérable` peut être une liste, par exemple, ou encore une chaîne de caractères, ou généralement tout objet ayant l'attribut `'__iter'`, c'est-à-dire sur lequel on peut itérer.

Exemple

- Écrire un programme qui affiche 20 fois « `J'aime la tartiflette` ». On peut générer un itérable d'entiers allant de 0 à `N-1` via `range(N)`, ou de façon équivalente `range(0,N)`.
- Écrire un programme qui compte de 1 à 10.
- Écrire un programme qui ajoute les inverses des entiers de `1` à `N`, où `N` est un entier demandé à l'utilisateur.

Boucles
Boucle conditionnelle

Sous Python, elle est implémentée comme suit :

```
while (condition) :
    < traitement >
```

où `condition` est un booléen. La boucle s'exécute tant que le booléen est vrai. Pour forcer la sortie de boucle, on peut utiliser l'instruction `break` au sein du traitement.

Exemple

Écrire un programme qui demande un entier à l'utilisateur tant que celui-ci n'est pas égal à 42.