Atelier d'informatique

Épisode II : Structures de données

26 février 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.

Structures conditionnelles

Sémantique

On peut demander à un programme de traiter différemment ses données selon les valeurs des variables introduites. Pour cela, on utilise une structure conditionnelle, qui prend la forme :

```
if (condition):
    < instructions >
else:
    < instructions >
```

le `else` étant optionnel si on ne désire rien faire si la condition n'est pas vérifiée.

Structures conditionnelles

Sémantique

On peut demander à un programme de traiter différemment ses données selon les valeurs des variables introduites. Pour cela, on utilise une structure conditionnelle, qui prend la forme :

```
if (condition):
    < instructions >
else:
    < instructions >
```

le `else` étant optionnel si on ne désire rien faire si la condition n'est pas vérifiée.
La condition est une expression de type `bool` pour *booléen*, nommé après le mathématicien et logicien George Boole. Une variable de type `bool` prend deux valeurs : `True` (*Vrai*) et `False` (*Faux*).

Structures conditionnelles

Tests de comparaison

Parmi les façons de construire des conditions booléennes, les plus courantes sont celles qui comparent les variables entre elles. Pour cela, on utilise les tests logiques de la table suivante :

Structures conditionnelles

Tests de comparaison

Parmi les façons de construire des conditions booléennes, les plus courantes sont celles qui comparent les variables entre elles. Pour cela, on utilise les tests logiques de la table suivante :

Égalité	x==y
Différent	x!=y
Inférieur ou égal	x<=y
Inférieur strictement	x<y
Supérieur ou égal	x>=y
Supérieur strictement	x>y

Structures conditionnelles

Tests de comparaison

Parmi les façons de construire des conditions booléennes, les plus courantes sont celles qui comparent les variables entre elles. Pour cela, on utilise les tests logiques de la table suivante :

Égalité	`x==y`
Différent	`x!=y`
Inférieur ou égal	`x<=y`
Inférieur strictement	`x<y`
Supérieur ou égal	`x>=y`
Supérieur strictement	`x>y`

Remarque Les comparaisons de type égalité fonctionnent sur toutes les variables (sauf celles de type `None`, dans lequel cas on teste `is None`). Les comparaisons de type inférieur/supérieur fonctionnent sur les nombres, ainsi que sur les chaînes de caractères. Dans le premier cas, c'est l'ordre usuel sur les nombres réels, et dans le deuxième cas l'ordre lexicographique (l'ordre des mots dans un dictionnaire).

Exemple 1

Définissez une variable x égale à 3. Quel est le type de l'expression `x==3` ? Évaluez sa valeur. Que donne l'évaluation de `x==2` ? Celle de `x<=4` ? Celle de `x<3` ? Celle de `x!=2` ?

Exemple 2

Définissez une variable x contenant la chaîne de caractère `"Wolfgang"`. Évaluez les booléens x > `"Amadeus"` et x < `"Mozart"`. Comparez x et `"Wagner"`.

Structures conditionnelles

Opérations sur les booléens

On peut combiner deux booléens a et b pour faire des opérations logiques en utilisant les opérateurs suivants :

Structures conditionnelles

Opérations sur les booléens

On peut combiner deux booléens a et b pour faire des opérations logiques en utilisant les opérateurs suivants :

Négation	`not(a)`
Conjonction (« et »)	`a and b`
Disjonction (« ou »)	`a or b`

Structures conditionnelles

Opérations sur les booléens

On peut combiner deux booléens a et b pour faire des opérations logiques en utilisant les opérateurs suivants :

Négation	`not(a)`
Conjonction (« et »)	`a and b`
Disjonction (« ou »)	`a or b`

Exercice 1

Écrire un programme qui demande deux nombres x et y à l'utilisateur et affiche s'ils sont tous les deux strictement positifs (c'est-à-dire s'ils sont > 0).

Exercice 2

Écrire un programme qui demande à l'utilisateur deux nombres x et y, et affiche « `Maximum de x et y =  ` » suivi du maximum de x et y (le plus grand des deux).

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.

Exemple 3

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

Exemple 3

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 4

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 4

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

On peut même compter à rebours...

Exemple 5

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

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 6

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 6

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

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

Exemple 7

Copiez la liste `l` dans une autre variable `t`. Exécutez l'instruction `t[0] = "Hegel"`. 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 6

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

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

Exemple 7

Copiez la liste `l` dans une autre variable `t`. Exécutez l'instruction `t[0] = "Hegel"`. 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...

Enfin, on peut ajouter une valeur à une liste en utilisant la méthode append : `l.append(x)` ajoute la valeur `x` à la fin de la liste `l`.

Exemple 8

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

Enfin, on peut ajouter une valeur à une liste en utilisant la méthode append : `l.append(x)` ajoute la valeur `x` à la fin de la liste `l`.

Exemple 8

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

Si vous souhaitez insérer un élément `x` en position `i`, utilisez `l.insert(i,x)`.

Enfin, on peut ajouter une valeur à une liste en utilisant la méthode append : `l.append(x)` ajoute la valeur `x` à la fin de la liste `l`.

Exemple 8

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

Si vous souhaitez insérer un élément `x` en position `i`, utilisez `l.insert(i,x)`.
Si vous voulez étendre la liste `l1` avec les valeurs d'une deuxième liste `l2`, utilisez `l1.extend(l2)`.

Enfin, on peut ajouter une valeur à une liste en utilisant la méthode append : `l.append(x)` ajoute la valeur `x` à la fin de la liste `l`.

Exemple 8

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

Si vous souhaitez insérer un élément `x` en position `i`, utilisez `l.insert(i,x)`.
Si vous voulez étendre la liste `l1` avec les valeurs d'une deuxième liste `l2`, utilisez `l1.extend(l2)`.
La méthode `l.sort()` trie la liste, la méthode `l.reverse()` renverse son ordre.

Listes dans Python

Slicing

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

Listes dans Python

Slicing

Pour obtenir une « sous-liste », ou un *slicing*, composé 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:p]` pour n'en prendre que un sur `p`.

Listes dans Python

Slicing

Pour obtenir une « sous-liste », ou un *slicing*, composé 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:p]` pour n'en prendre que un sur `p`.
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 dans Python

Slicing

Pour obtenir une « sous-liste », ou un *slicing*, composé 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:p]` pour n'en prendre que un sur `p`.
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.
On peut modifier en un coup la tranche `[i:j:p]`, en l'affectant :

Exercice 3

- Définissez une liste `li` de dix entiers.

Listes dans Python

Slicing

Pour obtenir une « sous-liste », ou un *slicing*, composé 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:p]` pour n'en prendre que un sur `p`.
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.
On peut modifier en un coup la tranche `[i:j:p]`, en l'affectant :

Exercice 3

- Définissez une liste `li` de dix entiers.
- Extrayez-en la tranche `[2 :6]`. Affichez le résultat. Triez la tranche (via la méthode `nom_liste.sort()`), et remettez le résultat dans `li`. Affichez `li`.

Listes dans Python

Opérations sur les listes

On peut ajouter deux listes ensemble : `l1 + l2` produit la *concaténation* des listes `l1` et `l2` (les éléments de la première suivis de ceux de la seconde).

Exemple 9

Définissez deux listes `l1` et `l2`, affichez `l1 + l2`.

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 que l'on peut parcourir.

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 que l'on peut parcourir. Le plus souvent, pour parcourir l'ensemble $[\![0, N-1]\!]$ des entiers de 0 à $N-1$, on utilise `range(N)`. En général, `range(i,j)` correspond à l'ensemble $[\![i, j-1]\!]$ des entiers de i à $j-1$.

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 que l'on peut parcourir. Le plus souvent, pour parcourir l'ensemble $[\![0, N-1]\!]$ des entiers de 0 à $N-1$, on utilise `range(N)`. En général, `range(i,j)` correspond à l'ensemble $[\![i, j-1]\!]$ des entiers de i à $j-1$.

Exercice 4

- Écrire un programme qui affiche 20 fois "J'aime la tartiflette".

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 que l'on peut parcourir. Le plus souvent, pour parcourir l'ensemble $[\![0, N-1]\!]$ des entiers de 0 à $N-1$, on utilise `range(N)`. En général, `range(i,j)` correspond à l'ensemble $[\![i, j-1]\!]$ des entiers de i à $j-1$.

Exercice 4

- Écrire un programme qui affiche 20 fois "J'aime la tartiflette".
- 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 que l'on peut parcourir. Le plus souvent, pour parcourir l'ensemble $[\![0, N-1]\!]$ des entiers de 0 à $N-1$, on utilise `range(N)`. En général, `range(i,j)` correspond à l'ensemble $[\![i, j-1]\!]$ des entiers de i à $j-1$.

Exercice 4

- Écrire un programme qui affiche 20 fois "J'aime la tartiflette".
- 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.

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.

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.

Exercice 5

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