Notion de listes

Cours

Sommaire

I La génération d'une liste

La liste est une structure de données en Python. Elle peut être générée de plusieurs manières : par extension, concaténation, duplication, ajouts successifs et la génération en compréhension.

A Définition et notation

Regrouper plusieurs informations dans une même structure de donnée est très utile en programmation. On introduit une première structure de données en Python, celle de liste.

DÉFINITION

Liste

Une liste est un objet algorithmique servant à stocker plusieurs variables au cours d'un programme. Les variables stockées peuvent être de différents types : variables numériques, chaînes de caractères, booléens, listes, etc. Les différents types de variables peuvent être mélangés au sein d'une même liste.

EXEMPLE

Instruction Python	Rôle de l'instruction
`liste1=[]`	définit la liste `liste1` comme la liste vide
`liste2=[3, 4.5, x]`	définit la liste `liste2` comme la liste contenant l'entier 3, le flottant 4.5 et la variable x

`liste3=['Bonjour', -5, t]`	définit la liste `liste3` comme la liste contenant la chaîne de caractères 'Bonjour', l'entier -5 et la variable t
`liste4=[0, [1, 2], "essai", True]`	définit la liste `liste4` comme la liste contenant l'entier 0, la liste `[1, 2]`, la chaîne de caractères `essai` et le booléen `True`

PROPRIÉTÉ

Une liste peut être nommée comme on le souhaite. Les éléments d'une liste se trouvent entre des crochets : $[...]$. Les éléments sont séparés par des virgules.

EXEMPLE

```
liste1=[]
```

```
liste2=[3, 4.5, x]
```

```
liste3=['Bonjour', -5, t]
```

```
liste4=[0, [1, 2], "essai", True]
```

B Les cinq modes de génération d'une liste

Il existe 5 mode de génération d'une liste : la génération d'une liste par extension, concaténation, duplication, ajouts successifs et en compréhension. Avec la méthode de génération en compréhension, on génère une liste en explicitant la relation mathématique qui existe entre chacun des termes.

PROPRIÉTÉ

On peut étendre une liste avec une autre liste en ajoutant les éléments d'une deuxième liste à la fin d'une première liste. En Python, on utilise la méthode `.extend()`.

EXEMPLE

Le programme Python suivant crée deux listes `liste1` et `liste2`, puis étend la liste `liste1` avec les éléments de la liste `liste2` :

```
liste1=[1,2,3]
liste2=[10,20,30]
liste1.extend(liste2) #étend liste1 avec les éléments de liste2
print(liste1)
```

On obtient : `[1, 2, 3, 10, 20, 30]` .

PROPRIÉTÉ

Un mode de génération de listes proche de l'extension est celui de la concaténation. Il a exactement le même effet que le précédent. La différence réside dans le fait de ne pas utiliser de méthode mais une "opération".

EXEMPLE

Le programme suivant donne le même résultat que le précédent mais en utilisant la concaténation :

```
liste1=[1,2,3]
liste2=[10,20,30]
liste1=liste1+liste2 #concaténation de liste1 avec liste2
print(liste1)
```

PROPRIÉTÉ

Un troisième mode de génération permettant le « recopiage » d'une liste au bout d'elle-même est la duplication. On peut concaténer la liste avec elle-même plusieurs fois de suite.

EXEMPLE

Le programme Python suivant duplique 5 fois `liste1` pour n'en faire qu'une liste :

```
liste1=[1,2,3]
liste1=liste1*5 #duplique 5 fois liste1
print(liste1)
```

On obtient : `[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]` .

PROPRIÉTÉ

On peut également générer une liste par ajouts successifs d'éléments. En Python, on utilise la méthode `.append()` . Elle sert à ajouter un élément à la fin d'une liste.

EXEMPLE

Le programme Python suivant crée une liste `liste1` vide, puis ajoute les carrés des entiers de 0 à 10 :

```
liste1=[]
for x in range (11): #pour x allant de 0 à 10
    liste1.append(x**2)#ajout du carré de x à la fin de liste1
print(liste1)
```

On obtient l'affichage suivant :

$[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]$.

PROPRIÉTÉ

Une autre façon de générer une liste est celle dite en compréhension. Elle est plus courte et plus explicite que les précédentes. Certains langages de programmation, comme Python, permettent de définir une liste en indiquant la façon dont sont générés les éléments de la liste.

EXEMPLE

Le programme Python suivant crée la liste des carrés des entiers de 0 à 10 d'une autre façon :

```
liste1=[x**2 for x in range(11)]
print(liste1)
```

On obtient l'affichage suivant :

$[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]$.

REMARQUE

On peut ajouter des conditions dans une liste créée en compréhension.

EXEMPLE

Le programme suivant crée la liste des carrés des entiers de 0 à 10 mais ne conserve que les résultats strictement supérieurs à 50 :

```
liste1=[x**2 for x in range(11) if x**2>50]
print(liste1)
```

On obtient : `[64, 81, 100]` .

REMARQUE

Si on utilise juste l'instruction `range(11)` , Python ne crée pas une liste. Pour générer la liste des entiers de 0 à 10 (inclus), on doit utiliser l'instruction `list` comme suit :

`list(range(11))` .

II La manipulation des éléments d'une liste

Il y a différentes manipulations possibles sur une liste : la suppression, l'insertion, le rangement des valeurs, le renversement, etc.

DÉFINITION

Rang d'un élément

On appelle **rang d'un élément** dans une liste sa position dans la liste.

REMARQUE

En Python, le premier élément d'une liste est de rang 0. La numérotation des éléments démarre donc à 0.

EXEMPLE

Dans la liste Python `[5, 10, 15]` , on a :

- le nombre 5 est l'élément de rang 0,
- le nombre 10 est l'élément de rang 1,
- le nombre 15 est l'élément de rang 2.

DÉFINITION

Longueur d'une liste

On appelle **longueur d'une liste** le nombre d'éléments d'une liste.

REMARQUE

En Python, la longueur de la liste `liste1` s'obtient avec l'instruction :

`len(liste1)` .

EXEMPLE

La liste `[5, 10, 15]` a pour longueur 3.

REMARQUE

Manipuler les éléments d'une liste nécessite souvent d'être capable de prendre un élément de rang donné dans une liste. Les instructions suivante sont à connaître lorsque l'on écrit un programme Python :

Instruction Python	Rôle de l'instruction
`liste1[i]`	affiche l'élément de rang `i` de la liste `liste1`
`liste1[2:5]`	affiche les éléments de la liste `liste1` du rang 2 au rang 5 (non compris)
`liste1[-1]`	affiche dernier élément de la liste `liste1`
`liste1[-4:]`	affiche les 4 derniers éléments de la liste `liste1`
`liste1[::3]`	renvoie une liste formée des termes de la liste `liste1` de rangs multiples de 3
`liste1.index(x)`	retrouve le rang de l'élément `x` dans la liste `liste1`
`liste1.count(a)`	compte le nombre d'occurrences de `a` dans la liste `liste1`
`a in liste1`	teste si l'élément `a` est dans la liste `liste1` et renvoie `True` si c'est le cas et `False` sinon

REMARQUE

On peut manipuler les éléments d'une liste d'un grand nombre de façons.

EXEMPLE

Le tableau suivant montre quelques exemples en Python :

Instruction Python	Rôle de l'instruction
`liste1[i]=a`	remplace l'élément de rang `i` par `a`
`liste1.insert(i,x)`	insère l'élément `x` dans la liste `liste1` au rang `i`
`liste1.remove(x)`	supprime la première occurrence de `x` dans la liste `liste1`
`liste1.pop(i)` ou `del(liste1[i])`	supprime l'élément d'indice `i` dans la liste `liste1`
`liste1.sort()`	modifie la liste `liste1` en triant les éléments dans l'ordre croissant
`liste2=sorted(liste1)`	trie la liste `liste1` et stocke le résultat dans `liste2`. La liste `liste1` n'est pas modifiée.
`liste1.reverse()`	inverse les éléments de la liste `liste1` (ainsi que les autres listes qui lui sont égales)
`liste2=liste1[::-1]`	définit la liste `liste2` comme étant la liste `liste1` dans l'ordre inverse mais ne modifie pas `liste1`

PROPRIÉTÉ

On peut également effectuer des transformations entre chaînes de caractères et listes.

EXEMPLE

Le programme suivant transforme la chaîne `chaine` en une liste de chaînes de caractères en utilisant l'espace comme séparateur pour couper la chaîne :

```
chaine="Bonjour à tous"
liste1=chaine.split(" ")
print(liste1)
```

On obtient : `['Bonjour', 'à', 'tous']`.

EXEMPLE

Le programme suivant transforme la liste `liste1` en une chaîne de caractères en ajoutant un espace entre chaque élément de la liste `liste1` :

```
liste1=["Bonjour","à","tous","!"]
chaine=" ".join(liste1)
print(chaine)
```

On obtient : `Bonjour à tous !`.

III Les itérations sur les éléments d'une liste et le parcours d'une liste

Une fois une liste créée, il est souvent utile de pouvoir retrouver un de ses éléments. Pour cela, on a besoin de parcourir la liste. Ce parcours peut se faire en utilisant les indices des éléments de la liste ou « en compréhension ».

REMARQUE

On peut parcourir les éléments d'une liste en utilisant leurs rangs.

EXEMPLE

Le programme en Python suivant affiche les éléments de la liste `liste1` un par un :

```
liste1=[0,1,2,3,4,5,6]
for i in range(len(liste1)):
    print(liste1[i])
```

REMARQUE

On peut également parcourir une liste sans avoir recours aux rangs.

EXEMPLE

Le programme suivant affiche également les éléments de la liste `liste1` un par un :

```
liste1=[0,1,2,3,4,5,6]
for x in liste1:
    print(x)
```

REMARQUE

On peut également appliquer une fonction à tous les éléments d'une liste.

EXEMPLE

Le programme en Python suivant applique la fonction $f : x \mapsto 2x^2 + 3x + 1$ à tous les éléments de la liste `liste1` :

```
liste1=list(range(11))#crée la liste des entiers de 0 à 10

def f(x):
    return 2*x**2+3*x+1

liste2=list(map(f,liste1))#map permet d'appliquer f à tous
#les éléments de liste1
print(liste2)
```

On obtient : `[1, 6, 15, 28, 45, 66, 91, 120, 153, 190, 231]`.

REMARQUE

Une autre façon d'appliquer une fonction à tous les éléments d'une liste utilise une fonction anonyme avec l'instruction `lambda` :

```
liste1=list(range(11))

liste2=list(map(lambda x: 2*x**2+3*x+1,liste1))
print(liste2)
```

L'instruction `lambda` est utilisée ici pour créer une fonction qui n'est pas destinée à être réutilisée, lui donner un nom est alors inutile. La syntaxe est la suivante :

`lambda nom_variable:expression_image_par_la_fonction`.

EXEMPLE

Dans l'exemple précédent, la fonction anonyme utilisée est la fonction $x \mapsto 2x^2 + 3x + 1$.