

Cours 3 : Les listes ; approfondissements

BCPST - Lycée Fénélon

Les listes en python

Les listes

Exemples

Parcours de liste

Liste définies par compréhension

Les méthodes `append()` et `pop()`

Extraction de sous-liste

Déclaration d'une liste par extension

Une liste est créée à l'aide d'une affectation. Une **déclaration par extension** consiste à déclarer la liste en donnant tous ses éléments entre deux crochets [.]

Déclaration d'une liste par extension

Une liste est créée à l'aide d'une affectation. Une **déclaration par extension** consiste à déclarer la liste en donnant tous ses éléments entre deux crochets [.]

Exemple : l'instruction suivante crée une liste nommée L et ayant 4 éléments, dont 3 entiers et une chaîne de caractères :

```
>>> L = [1, 2, 3, 'toto']
```

Déclaration d'une liste par extension

Une liste est créée à l'aide d'une affectation. Une **déclaration par extension** consiste à déclarer la liste en donnant tous ses éléments entre deux crochets [.]

Exemple : l'instruction suivante crée une liste nommée L et ayant 4 éléments, dont 3 entiers et une chaîne de caractères :

```
>>> L = [1, 2, 3, 'toto']
```

```
>>> print(L)
[1, 2, 3, 'toto']
>>> type(L)
<class 'list'>
>>> len(L)
4
```

La fonction len(.) prend en argument une liste et retourne son nombre d'éléments.

Accès aux éléments d'une liste

```
>>> L = [1, 2, 3, 'toto']
```

Les éléments d'une liste L s'obtiennent grâce à leur **indice** placé entre crochets.

Attention :

le premier élément a pour indice 0, le dernier a pour indice `len(L)-1` !

```
>>> L[0]
```

```
1
```

```
>>> L[1]
```

```
2
```

```
>>> L[len(L) - 1]
```

```
'toto'
```

Accès aux éléments d'une liste

```
>>> L = [1, 2, 3, 'toto']
```

Les éléments d'une liste L s'obtiennent grâce à leur **indice** placé entre crochets.

Attention :

le premier élément a pour indice 0, le dernier a pour indice `len(L)-1` !

```
>>> L[0]
```

```
1
```

```
>>> L[1]
```

```
2
```

```
>>> L[len(L) - 1]
```

```
'toto'
```

 **Astuce.** Si l'on place entre crochet un indice négatif, Python lui ajoute `len(L)`. Ainsi on peut accéder au dernier élément grâce à l'indice `-1`, à l'avant-dernier grâce à l'indice `-2`, etc.

```
>>> L[-1]
```

```
'toto'
```

Calcul de la somme d'une liste numérique

Exemple : écrire une **fonction** somme(L) :

- prenant en **argument une liste numérique L**,
- qui **renvoie la somme de ses éléments**.

```
def somme(L):  
    S = 0  
    for k in range(len(L)):  
        S = S + L[k]  
    return S
```

Exemple d'utilisation :

```
>>> L = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
>>> somme(L)  
55
```

Calcul de la moyenne d'une liste numérique

Exemple : écrire une **fonction** moyenne(L) :

- prenant en argument une liste numérique L,
- qui renvoie la moyenne de ses éléments.

```
def moyenne(L):  
    S = 0  
    n = len(L)  
    for k in range(n):  
        S = S + L[k]  
    return S/n
```

Exemple d'utilisation :

```
>>> L = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
>>> moyenne(L)  
5.5
```

Recherche du maximum dans une liste numérique

Exemple : écrire une **fonction** maximum(L) :

- prenant en argument une liste numérique L,
- qui renvoie le maximum de ses éléments.

```
def maximum(L):  
    M = L[0]  
    for k in range(1, len(L)):  
        if M < L[k]:  
            M = L[k]  
    return M
```

Exemple d'utilisation :

```
>>> L = [1, 12, 23, 4, 15, 36, 17, 81, 19, 12]  
>>> maximum(L)  
81
```

Recherche d'un élément dans une liste

Exemple : écrire une **fonction** recherche(L,e) :

- prenant **en arguments** :
 - **une liste L**,
 - **une donnée e**,
- qui **renvoie le booléen** :
 - True si e est un **élément de la liste L**,
 - False si e n'est pas un **élément de la liste L**,

```
def recherche(L,e):  
    for k in range(len(L)):  
        if e == L[k]:  
            return True  
    return False
```

Exemple d'utilisation :

```
>>> L = [1, 12, 23, 4, 15, 36, 17, 81, 19, 12]  
>>> recherche(L, 36)  
True
```

Appartenance d'un élément à une liste : in

- La commande `in` permet de déterminer si un élément appartient ou non à une liste.

```
>>> L = [1, -3, 5, 17.0, 2]
>>> 1 in L
True
```

Appartenance d'un élément à une liste : in

- La commande `in` permet de déterminer si un élément appartient ou non à une liste.

```
>>> L = [1, -3, 5, 17.0, 2]
>>> 1 in L
True
>>> -1 in L
False
>>> -3.0 in L
True
>>> 17 in L
True
>>> 'toto' in L
False
```

Parcours d'une liste : `for Variable in Liste:`

- Avec en plus la commande `for` on peut faire parcourir à une variable les éléments d'une liste :

```
for e in L:  
    print(e)
```

On dit que **les listes sont itérables**.

Parcours d'une liste : for Variable in Liste:

- Avec en plus la commande `for` on peut faire parcourir à une variable les éléments d'une liste :

```
for e in L:  
    print(e)
```

On dit que **les listes sont itérables**.

```
1  
-3  
5  
17.0  
2
```

Parcours d'une liste : for Variable in Liste:

- Avec en plus la commande `for` on peut faire parcourir à une variable les éléments d'une liste :

```
for e in L:  
    print(e)
```

On dit que **les listes sont itérables**.

```
1  
-3  
5  
17.0  
2
```

- Le parcours d'une liste peut se faire en sens inverse en utilisant la fonction `reversed()` :

```
for e in reversed(L):  
    print(e, end = " : ")
```

Parcours d'une liste : for Variable in Liste:

- Avec en plus la commande for on peut faire parcourir à une variable les éléments d'une liste :

```
for e in L:  
    print(e)
```

On dit que **les listes sont itérables**.

```
1  
-3  
5  
17.0  
2
```

- Le parcours d'une liste peut se faire en sens inverse en utilisant la fonction reversed() :

```
for e in reversed(L):  
    print(e, end = " : ")
```

```
2 : 17.0 : 5 : -3 : 1 :
```

Exemple : somme et moyenne d'une liste numérique

- Somme des éléments d'une suite numérique :

```
def somme(L):  
    S = 0  
    for x in L:  
        S = S + x  
    return S
```

Exemple : somme et moyenne d'une liste numérique

- Somme des éléments d'une suite numérique :

```
def somme(L):  
    S = 0  
    for x in L:  
        S = S + x  
    return S
```

- Moyenne des éléments d'une suite numérique :

```
def moyenne(L):  
    S = 0  
    for x in L:  
        S = S + x  
    return S/len(L)
```

Exemple : Recherche dans une liste

- Recherche du maximum dans une liste numérique :

```
def maximum(L):  
    M = L[0]  
    for x in L:  
        if x > M  
            M = x  
    return M
```

Exemple : Recherche dans une liste

- Recherche du maximum dans une liste numérique :

```
def maximum(L):  
    M = L[0]  
    for x in L:  
        if x > M  
            M = x  
    return M
```

- Recherche d'un élément dans une liste :

```
def recherche(L,e):  
    for x in L:  
        if x == e:  
            return True  
    return False
```

Compréhension de liste

On peut définir une liste à l'aide des mots-clés `for` et `in` comme on définit un ensemble en mathématiques 'par compréhension' :

Compréhension de liste

On peut définir une liste à l'aide des mots-clés `for` et `in` comme on définit un ensemble en mathématiques 'par compréhension' :

`[f(k) for k in range(m,n)]` correspond à $\{f(k) \mid k \in \llbracket m, n - 1 \rrbracket\}$
`[f(x) for x in liste]` correspond à $\{f(x) \mid x \in \text{liste}\}$

Compréhension de liste

On peut définir une liste à l'aide des mots-clés `for` et `in` comme on définit un ensemble en mathématiques 'par compréhension' :

`[f(k) for k in range(m,n)]` correspond à $\{f(k) \mid k \in \llbracket m, n - 1 \rrbracket\}$
`[f(x) for x in liste]` correspond à $\{f(x) \mid x \in \text{liste}\}$

Exemple :

```
>>> L = [ k for k in range(10) ]      # Liste des entiers de 0 à 9
>>> print(L)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Compréhension de liste

On peut définir une liste à l'aide des mots-clés `for` et `in` comme on définit un ensemble en mathématiques 'par compréhension' :

`[f(k) for k in range(m,n)]` correspond à $\{f(k) \mid k \in \llbracket m, n - 1 \rrbracket\}$
`[f(x) for x in liste]` correspond à $\{f(x) \mid x \in \text{liste}\}$

Exemple :

```
>>> L = [ k for k in range(10) ]      # Liste des entiers de 0 à 9
>>> print(L)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> L2 = [x**2 for x in L]          # liste de leurs carrés
>>> print(L2)
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Compréhension de liste

On peut définir une liste à l'aide des mots-clés `for` et `in` comme on définit un ensemble en mathématiques 'par compréhension' :

`[f(k) for k in range(m,n)]` correspond à $\{f(k) \mid k \in \llbracket m, n-1 \rrbracket\}$
`[f(x) for x in liste]` correspond à $\{f(x) \mid x \in \text{liste}\}$

Exemple :

```
>>> L = [ k for k in range(10) ]      # Liste des entiers de 0 à 9
>>> print(L)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> L2 = [x**2 for x in L]          # liste de leurs carrés
>>> print(L2)
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Exemple : liste X de $N = 100$ points régulièrement espacés entre $a = 0$ et $b = 9$:

```
>>> a, b, N = 0, 9, 100
>>> X = [ a+k*(b-a)/(N-1) for k in range(N)]
>>> print(X)
[0.0, 0.090909090909091, 0.181818181818182, 0.272727272727273, ...
..., 8.72727272727273, 8.81818181818182, 8.90909090909091, 9]
```

Les listes sont mutables

Les listes sont **modifiables** (on dit plutôt **mutables**) : on peut modifier leurs éléments.

```
>>> L = [1, 2, 3, 'toto'] # une liste
```

Les listes sont mutables

Les listes sont **modifiables** (on dit plutôt **mutables**) : on peut modifier leurs éléments.

```
>>> L = [1, 2, 3, 'toto'] # une liste
```

On modifie un élément de la liste en lui affectant une nouvelle valeur (de n'importe quel type).

```
>>> L[0] = 'le début'  
>>> L[2] = [-1, -2, -3]  
>>> print(L)  
['le début', 2, [-1,- 2, -3], 'toto']
```

C'est le seul cas où dans une affectation, à gauche du symbole d'affectation = ne figure pas une variable.

Les listes sont mutables

Les listes sont **modifiables** (on dit plutôt **mutables**) : on peut modifier leurs éléments.

```
>>> L = [1, 2, 3, 'toto'] # une liste
```

On modifie un élément de la liste en lui affectant une nouvelle valeur (de n'importe quel type).

```
>>> L[0] = 'le début'  
>>> L[2] = [-1, -2, -3]  
>>> print(L)  
['le début', 2, [-1,- 2, -3], 'toto']
```

C'est le seul cas où dans une affectation, à gauche du symbole d'affectation = ne figure pas une variable.

```
>>> print(L[-1], L[-2])  
'toto' [-1, -2, -3]  
>>> print(L[-5]), L[4])  
IndexError: list index out of range
```

Un indice qui n'est pas dans $[-1, \text{len}(L) - 1]$ produit une erreur

La méthode `append()`

La méthode `append()` appliquée aux listes permet d'ajouter un élément en fin de liste :

```
>>> L = [ ] ; print(L)
[]
>>> L.append(1) ; print(L)
[1]
>>> L.append(2) ; print(L)
[1, 2]
```

La méthode `append()`

La **méthode** `append()` appliquée aux listes permet d'ajouter un élément en fin de liste :

```
>>> L = [ ] ; print(L)
[]
>>> L.append(1) ; print(L)
[1]
>>> L.append(2) ; print(L)
[1, 2]
```

Exemple : créer la liste des carrés des entiers compris entre 0 et 20 :

```
>>> ListeCarres = [ ]           # initialisation
>>> for i in range(21):
...     ListeCarres.append(i ** 2)   # actualisation
... 
```

La méthode `append()`

La **méthode** `append()` appliquée aux listes permet d'ajouter un élément en fin de liste :

```
>>> L = [ ] ; print(L)
[]
>>> L.append(1) ; print(L)
[1]
>>> L.append(2) ; print(L)
[1, 2]
```

Exemple : créer la liste des carrés des entiers compris entre 0 et 20 :

```
>>> ListeCarres = [ ]          # initialisation
>>> for i in range(21):
...     ListeCarres.append(i ** 2)    # actualisation
...
```

```
>>> print(ListeCarres)
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196,
225, 256, 289, 324, 361, 400]
```

La méthode pop()

La méthode pop effectue l'opération inverse : elle retire le dernier élément de la liste, et le renvoie :

```
>>> L = [1,2,3]
>>> L.pop()
3
>>> print(L)
[1,2]
```

La méthode pop()

La méthode pop effectue l'opération inverse : elle retire le dernier élément de la liste, et le renvoie :

```
>>> L = [1,2,3]
>>> L.pop()
3
>>> print(L)
[1,2]
>>> L.pop()
2
>>> print(L)
[1]
```

La méthode pop()

La méthode pop effectue l'opération inverse : elle retire le dernier élément de la liste, et le renvoie :

```
>>> L = [1,2,3]
>>> L.pop()
3
>>> print(L)
[1,2]
>>> L.pop()
2
>>> print(L)
[1]
>>> L.pop()
1
>>> print(L)
[]
```

La méthode `pop()`

La méthode `pop` effectue l'opération inverse : elle retire le dernier élément de la liste, et le renvoie :

```
>>> L = [1,2,3]
>>> L.pop()
3
>>> print(L)
[1,2]
>>> L.pop()
2
>>> print(L)
[1]
>>> L.pop()
1
>>> print(L)
[]
```

- Avec un argument, `L.pop(i)` retire et renvoie l'élément `L[i]`, d'indice `i`.

Extraction de sous-liste

```
>>> L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
```

Extraction de sous-liste

```
>>> L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
```

```
>>> L1 = L[2:5]    # extraction de la sous-liste
```

```
>>> L1
```

```
['c', 'd', 'e']
```

L'instruction `L1 = L[2:5]` crée une nouvelle liste `L1` dont les éléments sont ceux de `L` allant de l'indice 2 (inclus) à l'indice 5 (exclu). On l'appelle une sous-liste extraite de la liste.

Extraction de sous-liste

```
>>> L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
```

```
>>> L1 = L[2:5]    # extraction de la sous-liste
```

```
>>> L1
```

```
['c', 'd', 'e']
```

L'instruction `L1 = L[2:5]` crée une nouvelle liste `L1` dont les éléments sont ceux de `L` allant de l'indice 2 (inclus) à l'indice 5 (exclu). On l'appelle une sous-liste extraite de la liste.

Les indices entre crochets peuvent sortir de la plage d'indice de la liste :

```
>>> L2 = L[0:100]    # tranche des indices de 0 à 100
```

```
>>> L2
```

```
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
```

Extraction de sous-liste

```
>>> L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
```

```
>>> L1 = L[2:5]    # extraction de la sous-liste  
>>> L1  
['c', 'd', 'e']
```

L'instruction `L1 = L[2:5]` crée une nouvelle liste `L1` dont les éléments sont ceux de `L` allant de l'indice 2 (inclus) à l'indice 5 (exclu). On l'appelle une sous-liste extraite de la liste.

Les indices entre crochets peuvent sortir de la plage d'indice de la liste :

```
>>> L2 = L[0:100]  # tranche des indices de 0 à 100  
>>> L2  
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
```

Un troisième paramètre définit un pas :

```
>>> L[6:2:-1]     # de 6 à 2 par pas de -1  
['g', 'f', 'e', 'd']
```

Extraction de sous-liste

LISTE[Index départ (inclus) : Indice arrivée (exclus) : Pas]

python crée la tranche en copiant l'élément de LISTE d'indice Indice de départ, puis tous les éléments obtenus en ajoutant successivement Pas à l'indice, tant qu'on ne dépasse pas Indice d'arrivée.

Extraction de sous-liste

LISTE[Index départ (inclus) : Indice arrivée (exclus) : Pas]

python crée la tranche en copiant l'élément de LISTE d'indice Indice de départ, puis tous les éléments obtenus en ajoutant successivement Pas à l'indice, tant qu'on ne dépasse pas Indice d'arrivée. **Par défaut : Pas = 1**

- ▶ si Pas > 0 : **par défaut** départ = 0 ; arrivée = len(LISTE)
- ▶ si Pas < 0 : **par défaut** départ = len(LISTE)-1 ; arrivée = -1

Extraction de sous-liste

LISTE[Index départ (inclus) : Indice arrivée (exclus) : Pas]

python crée la tranche en copiant l'élément de LISTE d'indice Indice de départ, puis tous les éléments obtenus en ajoutant successivement Pas à l'indice, tant qu'on ne dépasse pas Indice d'arrivée. **Par défaut** : Pas = 1

- ▶ si Pas > 0 : **par défaut** départ = 0 ; arrivée = len(LISTE)
- ▶ si Pas < 0 : **par défaut** départ = len(LISTE)-1 ; arrivée = -1

```
>>> L[::2]      # extraction par pas de 2
['a', 'c', 'e', 'g', 'i']
L[::-1]        # extraction par pas de -1
['i', 'h', 'g', 'f', 'e', 'd', 'c', 'b', 'a']
```

Extraction de sous-liste

LISTE[Index départ (inclus) : Indice arrivée (exclus) : Pas]

python crée la tranche en copiant l'élément de LISTE d'indice Indice de départ, puis tous les éléments obtenus en ajoutant successivement Pas à l'indice, tant qu'on ne dépasse pas Indice d'arrivée. **Par défaut** : Pas = 1

- ▶ si Pas > 0 : **par défaut** départ = 0 ; arrivée = len(LISTE)
- ▶ si Pas < 0 : **par défaut** départ = len(LISTE)-1 ; arrivée = -1

```
>>> L[::2]      # extraction par pas de 2
['a', 'c', 'e', 'g', 'i']
L[::-1]        # extraction par pas de -1
['i', 'h', 'g', 'f', 'e', 'd', 'c', 'b', 'a']
```

```
>>> L[3::2]     # départ de 3, par pas de 2
['d', 'f', 'h']
>>> L[:3:-1]   # de fin à 3 par pas de -1
['i', 'h', 'g', 'f', 'e']
```