Listes

Informatique pour tous

Liste

Une liste (de type **list**) en Python est un ensemble d'éléments ordonnés.

Pour créer une liste :

```
[element1, element2, ...]
```

Liste

```
In [1]: L = [1, 3, 9]

In [2]: L
Out[2]: [1, 3, 9]

In [3]: L = [0]

In [4]: L
Out[4]: [0]

In [5]: L = []

In [6]: L
Out[6]: []
```

Liste

Une liste peut contenir des types différents (et même une autre liste !) :

```
In [1]: liste = [3.14, "pi"]

In [2]: liste
Out[2]: [3.14, 'pi']

In [3]: L1 = [[1, 2], 3]

In [4]: L1
Out[4]: [[1, 2], 3]
```

Taille de liste

On peut connaître la taille (le nombre d'éléments) d'une liste en utilisant la fonction **len** :

```
In [9]: liste = [2.5, "blabla", True]

In [10]: len(liste)
Out[10]: 3
```

On peut aussi utiliser `len` sur les chaînes de caractères et les tuples.

Taille de liste

La seule liste de taille 0 est la liste vide :

```
In [31]: vide = []

In [32]: len(vide)
Out[32]: 0

In [33]: vide
Out[33]: []
```

Taille de liste

Question

Que renvoie `len([4, [7, 1], 2])` ?

Taille de liste

Question

Que renvoie `len([4, [7, 1], 2])` ?

Réponse : 3

Accéder à un élément

Chaque élément d'une liste a un **indice** : le premier est d'indice 0, le 2ème est d'indice 1...

Accéder à un élément

Chaque élément d'une liste a un **indice** : le premier est d'indice 0, le 2ème est d'indice 1...

`L[i]` donne l'élément d'indice `i` d'une liste `L`

Accéder à un élément

Chaque élément d'une liste a un **indice** : le premier est d'indice 0, le 2ème est d'indice 1...

`L[i]` donne l'élément d'indice `i` d'une liste `L`

⚠ On commence à compter à partir de 0 !
Le premier élément de `L` est `L[0]`, le dernier est

Accéder à un élément

Chaque élément d'une liste a un **indice** : le premier est d'indice 0, le 2ème est d'indice 1...

`L[i]` donne l'élément d'indice `i` d'une liste `L`

⚠ On commence à compter à partir de 0 !
Le premier élément de `L` est `L[0]`, le dernier est `L[len(L) - 1]`.

Accéder à un élément

`L[i]` donne l'élément d'indice `i` d'une liste `L` :

```
In [14]: L = [10, "abc", False, 3.14]

In [15]: L[0]
Out[15]: 10

In [16]: L[2]
Out[16]: False

In [17]: L[3]
Out[17]: 3.14
```

Accéder à un élément

⚠ Il ne faut pas essayer d'accéder à un élément qui n'existe pas !

Accéder à un élément

⚠ Il ne faut pas essayer d'accéder à un élément qui n'existe pas !

```
In [18]: L = [10, "abc", False, 3.14]

In [19]: L[4]
---------------------------------------
-----------------------
IndexError
most recent call last)
<ipython-input-19-ed6335ac1f62> in <m
----> 1 L[4]

IndexError: list index out of range
```

Accéder à un élément

Si on veut accéder au dernier élément d'une liste `L`, il y a un raccourci : `L[-1]`

Accéder à un élément

Si on veut accéder au dernier élément d'une liste `L`, il y a un raccourci : `L[-1]`

De même, on peut utiliser `L[-2]` pour accéder à l'avant dernier élément...

Accéder à un élément

Soit `L = [1, "a", -5, [4.23, 0, False], True]`

Question

Que vaut `L[1]` ?

Accéder à un élément

Soit `L = [1, "a", -5, [4.23, 0, False], True]`

Question

Que vaut `L[1]` ? `"a"`

Accéder à un élément

Soit `L = [1, "a", -5, [4.23, 0, False], True]`

Question

Que vaut `L[1]` ? `"a"`

Que vaut `L[5]` ?

Accéder à un élément

Soit `L = [1, "a", -5, [4.23, 0, False], True]`

Question

Que vaut `L[1]` ? `"a"`

Que vaut `L[5]` ? Erreur

Accéder à un élément

Soit `L = [1, "a", -5, [4.23, 0, False], True]`

Question

Que vaut `L[1]` ? `"a"`
Que vaut `L[5]` ? Erreur
Que vaut `L[-2]` ?

Accéder à un élément

Soit `L = [1, "a", -5, [4.23, 0, False], True]`

Question

Que vaut `L[1]` ? `"a"`
Que vaut `L[5]` ? Erreur
Que vaut `L[-2]` ? `[4.23, 0, False]`

Parcourir une liste

Il est possible de parcourir les indices d'une liste avec un `for`.
Par exemple, pour afficher le contenu d'une liste L :

```
for i in range(len(L)):
    print(L[i])
```

Remarque : on peut aussi simplement écrire `print(L)`

Recherche dans une liste

Pour savoir si une liste `L` contient un élément `e` :

Recherche dans une liste

Pour savoir si une liste `L` contient un élément `e` :

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

Recherche dans une liste

Pour savoir si une liste `L` contient un élément `e` :

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

⚠ `return False` est **après** le for, pas dedans !

Somme d'une liste

Question

Écrire une fonction qui prend en entrée une liste de nombres et qui renvoie la somme des éléments de cette liste.

Somme

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

Somme

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

```
In [15]: somme([3, -1, 7, 2])
Out[15]: 11
```

Question 5 On considère la fonction Python suivante :

```
def somme(L):
    S=0
    for i in range(len(L)-1):
        S=S+L[i]
        return S
```

Parmi les affirmations suivantes, indiquez celle ou celles qui sont vraies.

A) `somme([1,2,3,4])` renvoie 10.

B) `somme([1,2,3,4])` renvoie 6.

C) `somme([1,2,3,4])` renvoie un message d'erreur.

D) `S` est une variable locale.

Moyenne

Question

Écrire une fonction qui prend en entrée une liste de nombres et qui renvoie la moyenne de cette liste.

Moyenne

Question

Écrire une fonction qui prend en entrée une liste de nombres et qui renvoie la moyenne de cette liste.

```
def moyenne(L):
    res = somme(L)
    if len(L) != 0:
        res = res / len(L)
    return res
```

Moyenne

Question

Écrire une fonction qui prend en entrée une liste de nombres et qui renvoie la moyenne de cette liste.

```
def moyenne(L):
    res = somme(L)
    if len(L) != 0:
        res = res / len(L)
    return res
```

```
In [17]: moyenne([3, 5, 7, 9])
Out[17]: 6.0
```

Extraire une sous-liste

On peut extraire une sous-liste d'une liste `L` avec la syntaxe

`L[début:fin]`

```
In [3]: liste = [1, "bla", False, 7.3]

In [4]: sousListe = liste[1:3]

In [5]: sousListe
Out[5]: ['bla', False]
```

Extraire une sous-liste

On peut extraire une sous-liste d'une liste `L` avec la syntaxe

`L[début:fin]`

```
In [3]: liste = [1, "bla", False, 7.3]

In [4]: sousListe = liste[1:3]

In [5]: sousListe
Out[5]: ['bla', False]
```

⚠ Comme pour `range`, l'indice de fin est **exclu** !

Extraire une sous-liste

Les indices de début et de fin ne sont pas obligatoires.
Par défaut, le début est le premier élément et la fin le dernier.

Par exemple, `L[:3]` extrait la sous-liste de `L` de l'indice 0 jusqu'à l'indice 2.

Extraire une sous-liste

```
In [9]: L = [1, 3, -9, 2, 5]

In [10]: L[:3]
Out[10]: [1, 3, -9]

In [11]: L[3:]
Out[11]: [2, 5]

In [12]: L[:]
Out[12]: [1, 3, -9, 2, 5]
```

Chaine de caractères

Il est possible d'utiliser `[i]` sur les chaines de caractères et les tuples :

```
In [13]: s = "abcde"

In [14]: s[2]
Out[14]: 'c'

In [15]: s[1:4]
Out[15]: 'bcd'

In [16]: p = (1.3, 7.4)

In [17]: p[0]
Out[17]: 1.3

In [18]: p[1]
Out[18]: 7.4
```

Modifier une liste

Pour modifier l'élément d'indice `i` d'une liste `L` :

```
L[i] = ...
```

Modifier une liste

```
In [34]: L = [1, 8, 4, 3, 2]

In [35]: L[2] = 27

In [36]: L
Out[36]: [1, 8, 27, 3, 2]
```

str n'est pas modifiable

⚠ Il n'est pas possible de modifier une chaine de caractères :

```
In [3]: s = "abcde"

In [4]: s[2] = "f"
---------------------------------------------------------------
TypeError                                 Traceback (mos
<ipython-input-4-1ff09467581d> in <module>()
----> 1 s[2] = "f"

TypeError: 'str' object does not support item assignment
```

Affectation de liste

Si on écrit `L2 = L1` pour affecter une liste `L2` avec une liste `L1`, `L1` et `L2` sont la **même** liste : si l'on modifie l'une, on modifie aussi l'autre !

Affectation de liste

```
In [37]: L = [1, 4, 3.2]

In [38]: L1 = L

In [39]: L1 = [1, 4, 3.2]

In [40]: L2 = L1

In [41]: L1[1] = 42

In [42]: L1
Out[42]: [1, 42, 3.2]

In [43]: L2
Out[43]: [1, 42, 3.2]
```

Affectation de liste

Par contre, un slicing `L1[i:j]` effectue une copie, on peut donc avoir une copie indépendante de `L1` en écrivant `L2 = L1[:]`

```
In [6]: L2 = L1[:]

In [7]: L2[1] = 999

In [8]: L2
Out[8]: [4, 999, 2]

In [9]: L1
Out[9]: [4, 3.1, 2]
```

Affectation de liste

De la même façon, si on passe une liste à une fonction et qu'on la modifie, cela modifie la liste initiale !

Affectation de liste

De la même façon, si on passe une liste à une fonction et qu'on la modifie, cela modifie la liste initiale !

```
def f(L):
    L[0] = 1
```

Affectation de liste

De la même façon, si on passe une liste à une fonction et qu'on la modifie, cela modifie la liste initiale !

```
def f(L):
    L[0] = 1
```

```
In [2]: L1 =["blabla", True, False, 4.1]

In [3]: f(L1)

In [4]: L1
Out[4]: [1, True, False, 4.1]
```

Ajout d'un élément

Pour ajouter un élément `e` à une liste `L` :

```
L.append(e)
```

Ajout d'un élément

Pour ajouter un élément `e` à une liste `L` :

```
L.append(e)
```

Exemple :

```
In [2]: L = [1, "blabla", False]

In [3]: L.append(23)

In [4]: L
Out[4]: [1, 'blabla', False, 23]
```

Bien noter la syntaxe particulière...

Quelques exercices

Question

Écrire une fonction `echange` telle que `echange(L, i, j)` échange `L[i]` et `L[j]`.

Quelques exercices

Question

Écrire une fonction `echange` telle que `echange(L, i, j)` échange `L[i]` et `L[j]`.

```
def echange(L, i, j):
    L[i], L[j] = L[j], L[i]
```

Quelques exercices

Soit u_k la suite :

$$\begin{cases} u_0 = 5. \\ u_{k+1} = 2u_k + k, \quad \text{si } k \geq 0. \end{cases}$$

Question

Écrire une fonction **suite** qui prend un entier n en argument et renvoie la liste des termes de u_k pour $0 \leq k \leq n$.

Quelques exercices

```
def suite(n):
    L = [5]
    for k in range(n):
        L.append(L[-1]*2 + k)
    return L
```

```
In [7]: suite(5)
Out[7]: [5, 10, 21, 44, 91, 186]
```

Quelques exercices

Question

Écrire une fonction `inverse` qui prend une liste en argument et renvoie la liste inversée.

Quelques exercices

Question

Écrire une fonction `inverse` qui prend une liste en argument et renvoie la liste inversée.

Il faut créer une nouvelle liste qui va contenir la liste inversée.

Quelques exercices

```
def inverse(L):
    Linv = []
    for i in range(len(L)):
        Linv.append(L[-i - 1])
    return Linv
```

```
In [11]: inverse([1, 2, 3, 4])
Out[11]: [4, 3, 2, 1]
```

Quelques exercices

Question

Écrire une fonction `egal` qui teste si deux listes contiennent les mêmes éléments, dans le même ordre.