Les listes en python

Lycée Pierre Corneille – MP

2017-2018

Définitions

- **Listes** :

```
[1,2,3,4,5,6]          # entiers
[3.14, 0.693, 2.78] # flottants
[True, True, False] # booléens
['a', 'bra', 'ca', 'da', 'bra']
                       # chaînes de caractères
[1, 1.609, 'bonjour', None]
```

Définitions

- **Listes** :

```
[1,2,3,4,5,6]          # entiers
[3.14, 0.693, 2.78] # flottants
[True, True, False] # booléens
['a', 'bra', 'ca', 'da', 'bra']
                       # chaînes de caractères
[1, 1.609, 'bonjour', None]
```

- **Longueur** de la liste : `len(L)`

Définitions

- **Listes** :

```
[1,2,3,4,5,6]          # entiers
[3.14, 0.693, 2.78] # flottants
[True, True, False] # booléens
['a', 'bra', 'ca', 'da', 'bra']
                       # chaînes de caractères
[1, 1.609, 'bonjour', None]
```

- **Longueur** de la liste : `len(L)`
- **Indices valides** : $-N \leqslant k \leqslant N-1$

Définitions

- **Listes** :

```
[1,2,3,4,5,6]          # entiers
[3.14, 0.693, 2.78] # flottants
[True, True, False] # booléens
['a', 'bra', 'ca', 'da', 'bra']
                       # chaînes de caractères
[1, 1.609, 'bonjour', None]
```

- **Longueur** de la liste : `len(L)`
- **Indices valides** : $-N \leqslant k \leqslant N-1$
- Indice *non* valide $\Longrightarrow$ `IndexError: list index out of range`

Définitions

- **Listes** :

```
[1,2,3,4,5,6]          # entiers
[3.14, 0.693, 2.78] # flottants
[True, True, False] # booléens
['a', 'bra', 'ca', 'da', 'bra']
                       # chaînes de caractères
[1, 1.609, 'bonjour', None]
```

- **Longueur** de la liste : `len(L)`
- **Indices valides** : $-N \leqslant k \leqslant N-1$
- Indice *non* valide $\Longrightarrow$ `IndexError: list index out of range`
- **Liste vide** : `[]`

Création d'une liste

- Conversion d'un itérable

```
list(range(10))
list(range(1,11))
list(range(0,30,5))
list(range(0,-10,-1))
```

Création d'une liste

- Conversion d'un itérable

```
list(range(10))
list(range(1,11))
list(range(0,30,5))
list(range(0,-10,-1))
```

- Énumération par une formule globale

```
[2*n+1 for n in range(10)]
[sin(3*n) for n in range(10) if sin(3*n)>0]
```

Accès à un élément

$L = (L_0, \ldots, L_{N-1})$

- Si $0 \leqslant k < N$, alors `L[k]` $= L_k$.

Accès à un élément

$L = (L_0, \ldots, L_{N-1})$

- Si $0 \leqslant k < N$, alors `L[k]` $= L_k$.
- Si $-N \leqslant k < 0$, alors $0 \leqslant N + k < N$ et `L[k]==L[N+k]`.

Accès à un élément

Une liste est une **variable mutable**.

- ```
  L = [0,1,2,3,4,5,6]
  L[0] = 2015
  L
  ```
  ```

Accès à un élément

Une liste est une **variable mutable**.

- ```
  L = [0,1,2,3,4,5,6]
  L[0] = 2015
  L
  ```

- ```
  def f(L):
    L[0] = 2016
    return None

  f(L)
  L
  ```

Tranches

Liste $L = (L_0, \ldots, L_{N-1})$
Indices valides $0 \leqslant i \leqslant j$

- Si $i \geqslant 0$ et $j \geqslant 0$, alors `L[i:j]` $= (L_k)_{i \leqslant k < j}$.

Tranches

Liste $L = (L_0, \ldots, L_{N-1})$
Indices valides $0 \leqslant i \leqslant j$

- Si $i \geqslant 0$ et $j \geqslant 0$, alors `L[i:j]` $= (L_k)_{i \leqslant k < j}$.
- `L[i:]` $= (L_k)_{i \leqslant k} = (L_k)_{i \leqslant k < N}$

Tranches

Liste $L = (L_0, \ldots, L_{N-1})$
Indices valides $0 \leqslant i \leqslant j$

- Si $i \geqslant 0$ et $j \geqslant 0$, alors `L[i:j]` $= (L_k)_{i \leqslant k < j}$.
- `L[i:]` $= (L_k)_{i \leqslant k} = (L_k)_{i \leqslant k < N}$
- `L[:j]` $= (L_k)_{k < j} = (L_k)_{0 \leqslant k < j}$

Tranches

Liste $L = (L_0, \ldots, L_{N-1})$
Indices valides $0 \leqslant i \leqslant j$

- Si $i \geqslant 0$ et $j \geqslant 0$, alors `L[i:j]` $= (L_k)_{i \leqslant k < j}$.
- `L[i:]` $= (L_k)_{i \leqslant k} = (L_k)_{i \leqslant k < N}$
- `L[:j]` $= (L_k)_{k < j} = (L_k)_{0 \leqslant k < j}$
- La tranche `L[:]` est une **copie** de la liste `L`.

```
L = [0,1,2,3,4,5,6]
M = L[:]
L == M              # True
L[3] = 0
L == M              # False
```

Tranches

Liste $L = (L_0, \ldots, L_{N-1})$
Indices valides $i \leqslant j < 0$

- `L[i:j]` $= (L_k)_{N+i \leqslant k < N+j}$

Tranches

Liste $L = (L_0, \ldots, L_{N-1})$
Indices valides $i \leqslant j < 0$

- `L[i:j]` $= (L_k)_{N+i \leqslant k < N+j}$
- `L[i:]` $= (L_k)_{N+i \leqslant k < N}$

Tranches

Liste $L = (L_0, \ldots, L_{N-1})$
Indices valides $i \leqslant j < 0$

- `L[i:j]` $= (L_k)_{N+i \leqslant k < N+j}$
- `L[i:]` $= (L_k)_{N+i \leqslant k < N}$
- `L[:j]` $= (L_k)_{0 \leqslant k < N+j}$

Tranches

Liste $L = (L_0, \ldots, L_{N-1})$
Indices valides $i \leqslant j < 0$

- `L[i:j]` $= (L_k)_{N+i \leqslant k < N+j}$
- `L[i:]` $= (L_k)_{N+i \leqslant k < N}$
- `L[:j]` $= (L_k)_{0 \leqslant k < N+j}$

Indices valides i, j

Tranches

Liste $L = (L_0, \ldots, L_{N-1})$
Indices valides $i \leqslant j < 0$

- `L[i:j]` $= (L_k)_{N+i \leqslant k < N+j}$
- `L[i:]` $= (L_k)_{N+i \leqslant k < N}$
- `L[:j]` $= (L_k)_{0 \leqslant k < N+j}$

Indices valides i, j

- Si $i < 0 \leqslant j$, alors `L[i:j]` $= (L_k)_{N+i \leqslant k < j}$.

Tranches

Liste $L = (L_0, \ldots, L_{N-1})$
Indices valides $i \leqslant j < 0$

- `L[i:j]` $= (L_k)_{N+i \leqslant k < N+j}$
- `L[i:]` $= (L_k)_{N+i \leqslant k < N}$
- `L[:j]` $= (L_k)_{0 \leqslant k < N+j}$

Indices valides i, j

- Si $i < 0 \leqslant j$, alors `L[i:j]` $= (L_k)_{N+i \leqslant k < j}$.
- Si $j < 0 \leqslant i$, alors `L[i:j]` $= (L_k)_{i \leqslant k < N+j}$.

Tranches

- Si $p > 0$, alors

$$\texttt{L[i:j:p]} = (L_k)_{[i \leqslant k < j] \cap [k = i \mod p]}$$

Tranches

- Si $p > 0$, alors

$$\texttt{L[i:j:p]} = (L_k)_{[i \leqslant k < j] \cap [k = i \mod p]}$$

- Cas $p < 0$

Tranches

- Si $p > 0$, alors

$$\texttt{L[i:j:p]} = (L_k)_{[i \leqslant k < j] \cap [k = i \mod p]}$$

- Cas $p < 0$
- $\texttt{L[::-1]} = (L_{N-1}, L_{N-2}, \ldots, L_2, L_1, L_0)$

Recherche d'un élément

- **Appartenance** : `x in L`

Recherche d'un élément

- **Appartenance** : `x in L`
- **Nombre d'occurrences** : `L.count(x)`.

Recherche d'un élément

- **Appartenance** : `x in L`
- **Nombre d'occurrences** : `L.count(x)`.
- **Première occurrence** : `L.index(x)`

Recherche d'un élément

- **Appartenance** : `x in L`
- **Nombre d'occurrences** : `L.count(x)`.
- **Première occurrence** : `L.index(x)`
  `ValueError`

Insertion et suppression d'un élément

- `L.insert(i, x)`

Insertion et suppression d'un élément

- `L.insert(i, x)`
- `L.remove(x)`

Insertion et suppression d'un élément

- `L.insert(i, x)`
- `L.remove(x)`

```
ValueError: list.remove(x): x not in list
```

Insertion et suppression d'un élément

- `L.insert(i, x)`
- `L.remove(x)`

  `ValueError: list.remove(x): x not in list`
- `L.pop(i)`

Insertion et suppression d'un élément

- `L.insert(i, x)`
- `L.remove(x)`

  `ValueError: list.remove(x): x not in list`
- `L.pop(i)`

  `L.pop()`

Insertion et suppression d'un élément

- `L.insert(i, x)`
- `L.remove(x)`
  `ValueError: list.remove(x): x not in list`
- `L.pop(i)`
  `L.pop()`

```
from random import randint
# On lance un dé 10 fois de suite :
L = [randint(1,6) for i in range(10)]
while (L!=[]):
  x = L.pop()
  print(x, L)
```

Concaténation

- **Augmentation** élémentaire : append

```
La = [1,2,3]
La.append(7)
```

Concaténation

- **Augmentation** élémentaire : append

```
La = [1,2,3]
La.append(7)
```

- **Concaténation** : +

```
Lb = [4,5,6]
Lb = Lb + [8]
La + Lb
```

Concaténation

- **Augmentation** élémentaire : `append`

```
La = [1,2,3]
La.append(7)
```

- **Concaténation** : `+`

```
Lb = [4,5,6]
Lb = Lb + [8]
La + Lb
```

- **Augmentation** : `extend`

```
La.extend(Lb)
```

Concaténation

- **Augmentation** élémentaire : `append`

```
La = [1,2,3]
La.append(7)
```

- **Concaténation** : `+`

```
Lb = [4,5,6]
Lb = Lb + [8]
La + Lb
```

- **Augmentation** : `extend`

```
La.extend(Lb)
```

- **Concaténations répétées** : `*`

```
3*Lb
Lb*3
```

Classement

- Inversion de l'ordre des éléments

```
La = [1,4,7,8,5,2]
Lb = La[::-1]
La.reverse()
La == Lb
```

Classement

- Inversion de l'ordre des éléments

```
La = [1,4,7,8,5,2]
Lb = La[::-1]
La.reverse()
La == Lb
```

- Tri par ordre croissant : `L.sort()`.

Classement

- Inversion de l'ordre des éléments

```
La = [1,4,7,8,5,2]
Lb = La[::-1]
La.reverse()
La == Lb
```

- Tri par ordre croissant : `L.sort()`.
- Tri par ordre décroissant : `L.sort(reverse=True)`.

Classement

- Inversion de l'ordre des éléments

```
La = [1,4,7,8,5,2]
Lb = La[::-1]
La.reverse()
La == Lb
```

- Tri par ordre croissant : `L.sort()`.
- Tri par ordre décroissant : `L.sort(reverse=True)`.
- **Copie triée** par ordre croissant : `sorted(L)`

Opérations algébriques

- Listes de nombres ou de chaînes de caractères
  - `min(L)`

Opérations algébriques

- Listes de nombres ou de chaînes de caractères
  - `min(L)`
  - `max(L)`

Opérations algébriques

- Listes de nombres ou de chaînes de caractères
  - `min(L)`
  - `max(L)`
- Liste de *nombres* : `sum(L)`.