Structures de données et pile

Informatique pour tous

Éléments d'une structure de données

Structure de donnée = moyen de stocker plusieurs éléments.

Éléments d'une structure de données

Structure de donnée = moyen de stocker plusieurs éléments.

Quelques structures de données que vous avez vu en première année:

- Chaîne de caractères (`str`)
- N-uplets (`tuple`)
- Listes (`list`)
- Tableau (`array`)

Éléments d'une structure de données

Une structure de données peut contenir:

- `str`: que des **caractères**.
- `tuple`: **n'importe quels** types.
- `list`: **n'importe quels** types.
- `array`: que des objets du **même type**.

Taille

Il est possible de connaître la taille d'une structure en **temps constant** $O(1)$ avec la fonction len:

```
>>> len( np.array([1, 2, 3]) )
3

>>> len( (False, "aaa") )
2
```

Accéder

On peut accéder à un élément d'une structure de données avec [ ], en **une seule opération**.
Exemple:

```
>>> (3, 2)[1]
2

>>> [3, 4.2, "blabla", True][2]
'blabla'
```

Appartenance

Pour une structure de donnée de taille n, on peut tester si un objet y appartient en **temps linéaire** $O(n)$ avec **in**:

```
>>> False in [1, "aa", False, True]
True
```

Appartenance

Pour une structure de donnée de taille n, on peut tester si un objet y appartient en **temps linéaire** $O(n)$ avec **in**:

```
>>> False in [1, "aa", False, True]
True
```

Ce **in** est différent de celui utilisé dans **for** i **in** ...

Calcul vectoriel

On peut directement effectuer des opérations (+, -, *, ...) sur des tableaux (ce qui est plus rapide).
Les tableaux sont bien adaptés si vous voulez faire des calculs vectoriels ou matriciels.

Calcul vectoriel

On peut directement effectuer des opérations (+, -, *, ...) sur des tableaux (ce qui est plus rapide).
Les tableaux sont bien adaptés si vous voulez faire des calculs vectoriels ou matriciels.

Pour les list, tuple et str, le + correspond à la concaténation!

Mutable

Définition

Les structures dont on peut modifier un élément sont appelées **mutables**.

- `str`: non mutable.
- `tuple`: non mutable.
- `list`: mutable.
- `array`: mutable.

Mutable

Pour les types mutables `list` et `array`, on peut modifier un élément en **une opération**.

Mutable

`list` et `array` sont mutables:

```
In [23]: T = np.array([1, 2, 3])

In [24]: T[1] = 4

In [25]: T
Out[25]: array([1, 4, 3])
```

Non mutable

`str` et `tuple` ne sont pas mutables:

```
In [6]: a = (1, 2, 3)

In [7]: a[2] = 4
---------------------------------------------------------------
-----------------
TypeError                                 Traceback (most
recent call last)
<ipython-input-7-fdddcd2211ce> in <module>()
----> 1 a[2] = 4

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

Dynamique vs statique

Définition

Une structure auquel on peut supprimer/ajouter des éléments est dite **dynamique**.
Sinon, elle est dite **statique**.

- `str`: statique.
- `tuple`: statique.
- `list`: dynamique.
- `array`: statique.

Suppression

Il est possible de supprimer le `i`ème élément d'une liste `L` en utilisant `L.pop(i)`, qui renvoie l'élément supprimé.

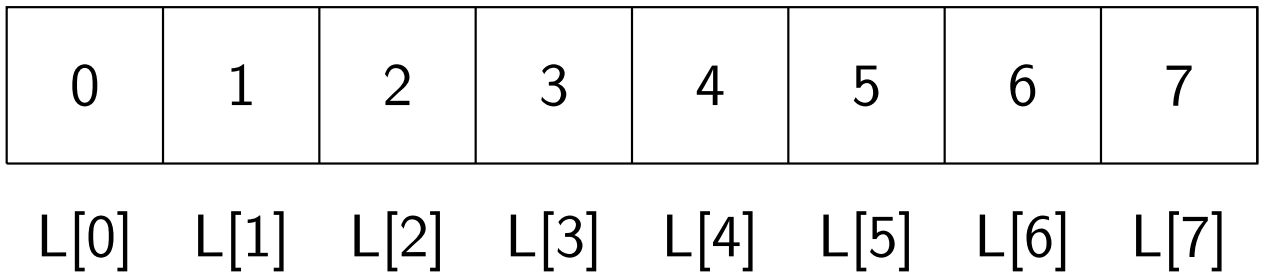
Suppression

Il est possible de supprimer le `i`ème élément d'une liste `L` en utilisant `L.pop(i)`, qui renvoie l'élément supprimé.
`L.pop()` supprime et renvoie le **dernier élément**.

```
In [1]: L = [2, 3, 5, 7]

In [2]: L.pop(1)
Out[2]: 3

In [3]: L
Out[3]: [2, 5, 7]

In [4]: L.pop()
Out[4]: 7

In [5]: L
Out[5]: [2, 5]
```

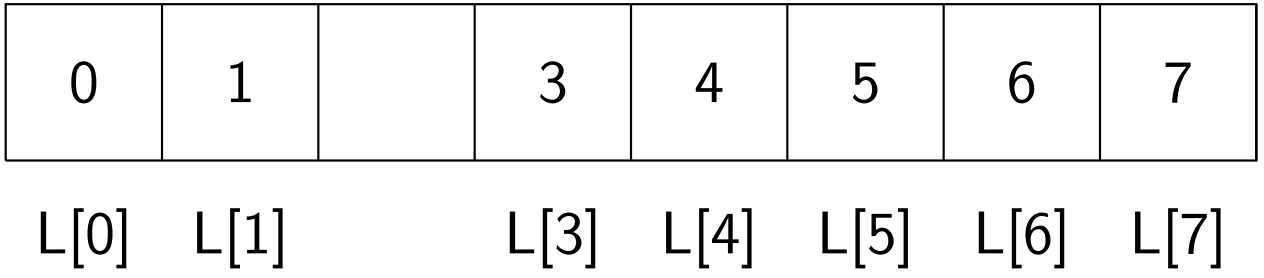
pop

Imaginons que L soit la liste suivante:



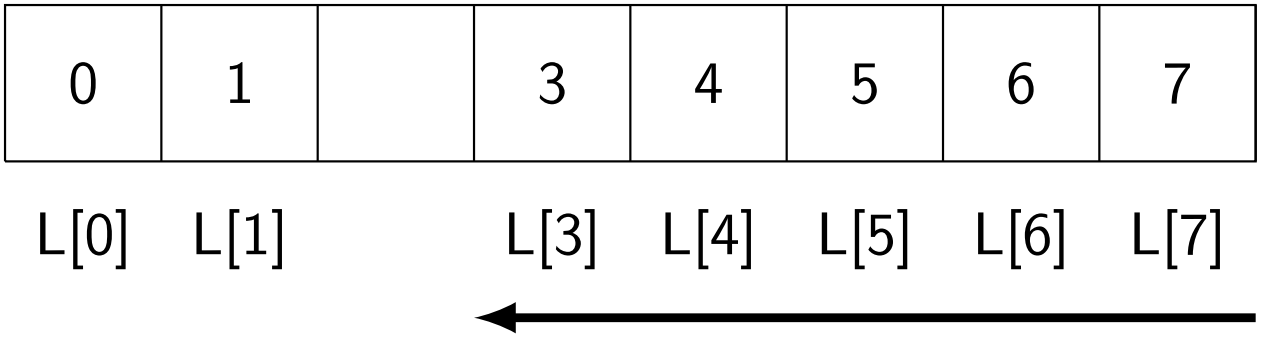
pop

On appelle `L.pop(2)`:



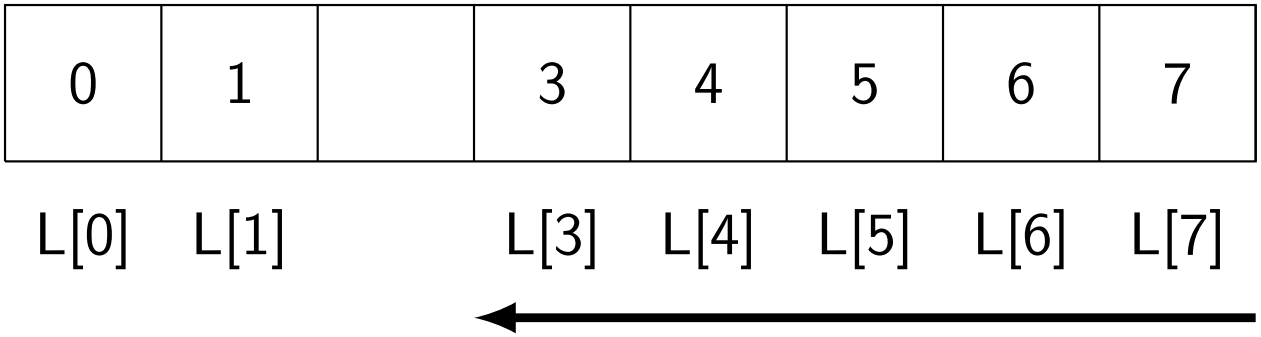
pop

On appelle `L.pop(2)`:



pop

Il faut déplacer 3 en $L[2]$, puis 4 en $L[3]$, ...



pop

Dans le pire des cas, `L.pop(i)` nécessite donc `len(L)` opérations!

pop

Dans le pire des cas, `L.pop(i)` nécessite donc `len(L)` opérations!

Cependant `L.pop()` nécessite une seule opération.

Pile

De façon abstraite, une pile est une structure de données dynamique possédant:

- Une fonction `append`: ajoute un élément à la fin.
- Une fonction `empty`: renvoie **True** si la pile est vide, **False** sinon.
- Une fonction `pop`: supprime et renvoie le dernier élément.
  (Erreur si la pile est vide! Il faut tester avant si la pile est vide.)

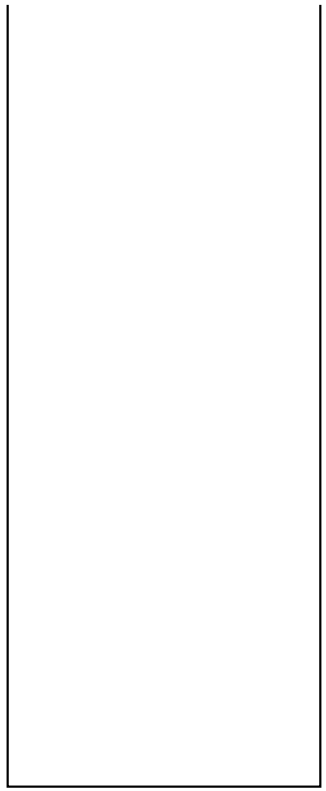
Pile

De façon abstraite, une pile est une structure de données dynamique possédant:

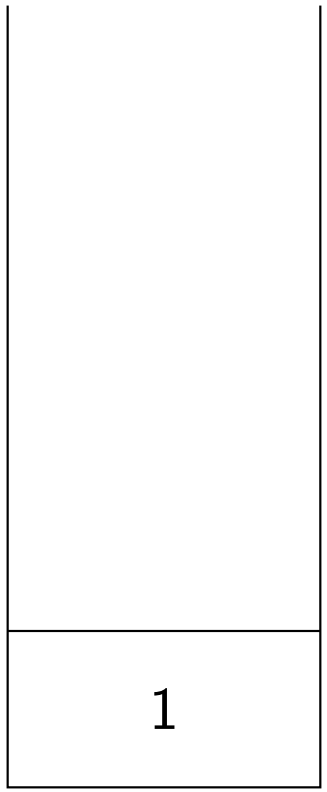
- Une fonction `append`: ajoute un élément à la fin.
- Une fonction `empty`: renvoie **True** si la pile est vide, **False** sinon.
- Une fonction `pop`: supprime et renvoie le dernier élément.
  (Erreur si la pile est vide! Il faut tester avant si la pile est vide.)

Comme une pile d'assiettes: on pose et retire une assiette seulement « en haut ».

Pile

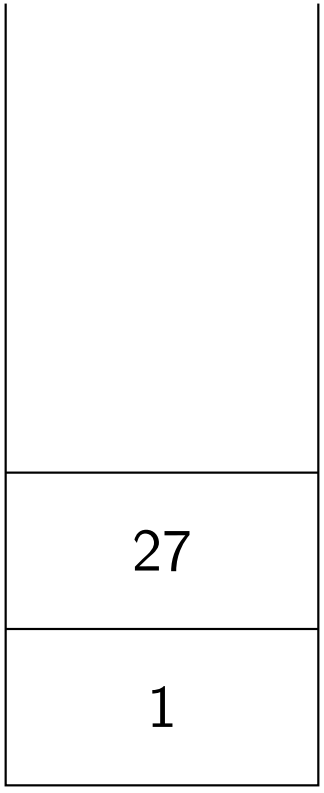


Pile



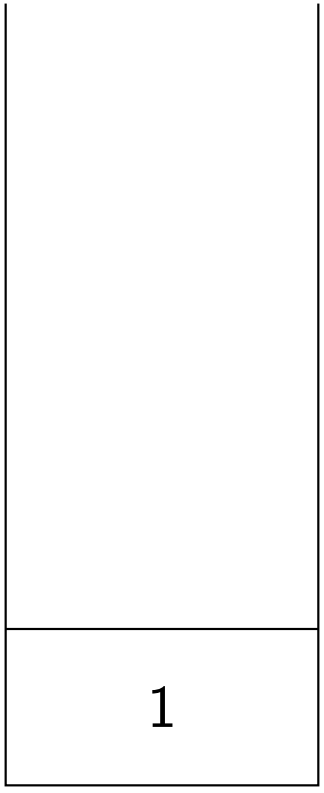
append(1)

Pile



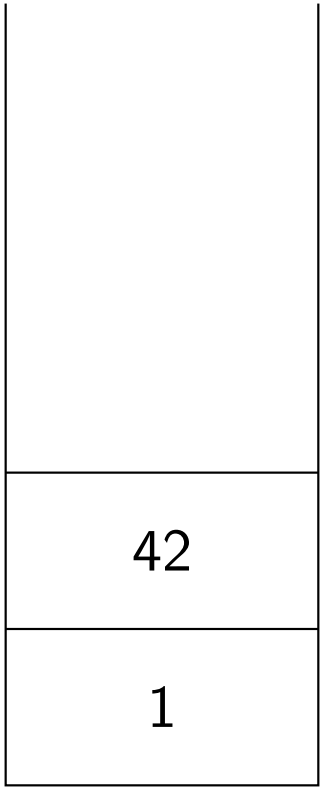
append(27)

Pile



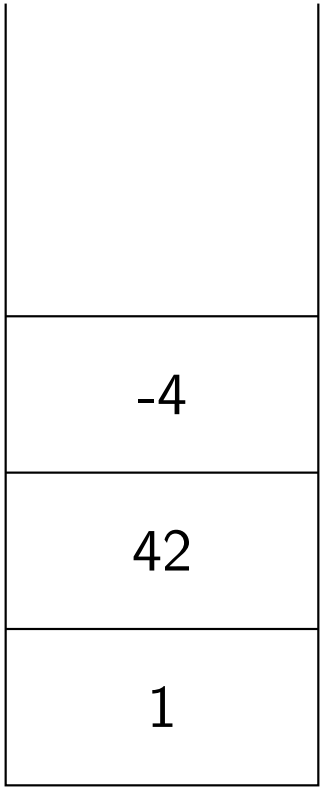
pop(): retourne 27

Pile



append(42)

Pile



append(-4)

Pile

Une liste possède des fonctions `pop`, `append` et `len` qui permet de savoir si une `list` est vide.

Pile

Une liste possède des fonctions `pop`, `append` et `len` qui permet de savoir si une `list` est vide.

Donc on peut se servir d'une liste comme d'une pile.

Mots bien parenthésés

Soit s une chaîne de caractères composée uniquement de parenthèses.

Définition

On dit que s est **bien parenthésée** si, en parcourant s de gauche à droite, le nombre de parenthèses ouvrantes est toujours supérieur au nombre de parenthèses fermantes et les nombres totaux de parenthèses ouvrantes et fermantes sont les même.

Mots bien parenthésés

Soit s une chaîne de caractères composée uniquement de parenthèses.

Définition

On dit que s est **bien parenthésée** si, en parcourant s de gauche à droite, le nombre de parenthèses ouvrantes est toujours supérieur au nombre de parenthèses fermantes et les nombres totaux de parenthèses ouvrantes et fermantes sont les même.

Exemple: "()" et "(()())" sont bien parenthésées.
")(" et "(()))" sont mal parenthésées.

Mots bien parenthésés

Exercice

Écrire un algorithme pour savoir si une suite de caractères s est bien parenthésée.

Mots bien parenthésés

Exercice

Écrire un algorithme pour savoir si une suite de caractères s est bien parenthésée.

On définit une pile vide.

Mots bien parenthésés

Exercice

Écrire un algorithme pour savoir si une suite de caractères s est bien parenthésée.

On définit une pile vide.
En parcourant s de gauche à droite:

- Si on voit une parenthèse ouvrante, on l'ajoute sur la pile.

Mots bien parenthésés

Exercice

Écrire un algorithme pour savoir si une suite de caractères s est bien parenthésée.

On définit une pile vide.
En parcourant s de gauche à droite:

- Si on voit une parenthèse ouvrante, on l'ajoute sur la pile.
- Si on voit une parenthèse fermante: soit la pile est vide, ce qui veut dire qu'il y a eu plus de parenthèses fermantes qu'ouvrantes, soit on supprime le dessus de la pile.

Mots bien parenthésés

Exercice

Écrire un algorithme pour savoir si une suite de caractères s est bien parenthésée.

On définit une pile vide.
En parcourant s de gauche à droite:

- Si on voit une parenthèse ouvrante, on l'ajoute sur la pile.
- Si on voit une parenthèse fermante: soit la pile est vide, ce qui veut dire qu'il y a eu plus de parenthèses fermantes qu'ouvrantes, soit on supprime le dessus de la pile.

À la fin, il faut vérifier que la pile est vide (c'est à dire qu'il n'y ait pas de parenthèses ouvrantes en trop).

Mots bien parenthésés

```
def parenthese(s):
    pile = []
    for c in s:
        if c == "(":
            pile.append("(")
        else:
            if len(pile) == 0:
                return False
            pile.pop()
    return len(pile) == 0
```

Mots bien parenthésés

```
>>> s = "(()()))("
```

Mots bien parenthésés

```
>>> s = "(()()))("
```

```
>>> parenthese(s)
False
```

Mots bien parenthésés

```
>>> s = "(()()))("
```

```
>>> parenthese(s)
False
```

```
In [24]: parenthese("(()))()")
Out[24]: True
```

Question

Question 10 On considère la fonction Python suivante (où `P` est supposée être une pile non vide de nombres entiers) :

```
def maxi(P):
    m=P.pop()
    while len(P)>0:
        if m<P.pop():
            m=P.pop()
    return m
```

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

A) `maxi([3,2,1])` renvoie 3.

B) `maxi([1,3,2])` renvoie 3.

C) `max(P)` renvoie le maximum des nombres de la pile `P`.

D) `max([2,1])` provoque une erreur.