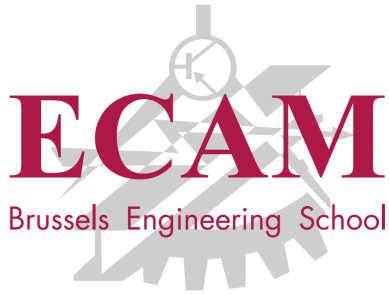


IC1T Programmation

Cours 5

Types abstraits de données et algorithme

Sébastien Combéfis, Quentin Lurkin

Rappels

- Définition de fonctions
  - divisions de problèmes
  - fonctions récursives
  - modules
- Séquences et listes
  - modifications de listes
  - parcours de listes
  - copies de listes

Objectifs

- Séquences
  - types abstraits de données
  - intervalles
  - itérateurs
- Algorithmes
  - spécification
  - tri

Types abstraits de données

Types abstraits de données

- Ensemble d'opérations sur une collection de données.
- Souvent basés sur des organisations usuelles d'objets.
- Très simples à réaliser avec les listes Python.

Piles

- Basé sur un empilement d'objets.

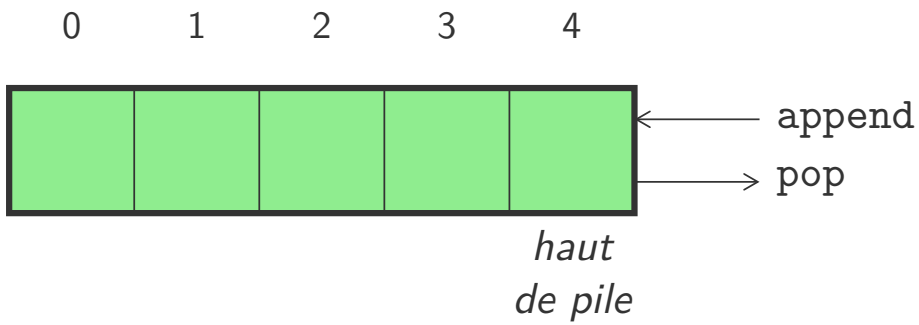
  pile d'assiettes par exemple.

- Accès facile au sommet de la pile.

- Pas accès aux autres éléments.

- Suit le principe LIFO

  Last-In First-Out, dernier élément ajouté sera premier à sortir



Piles

```
1  stack = []                    # la pile est vide
2  stack.append(1)               # la pile contient [1]
3  stack.append(2)               # la pile contient [1, 2]
4  stack.append(3)               # la pile contient [1, 2, 3]
5  result = stack.pop()          # la pile contient [1, 2]
6
7  print(result)
8  print(stack)
```

```
3
[1, 2]
```

Files

- Basé sur une file d'objets.

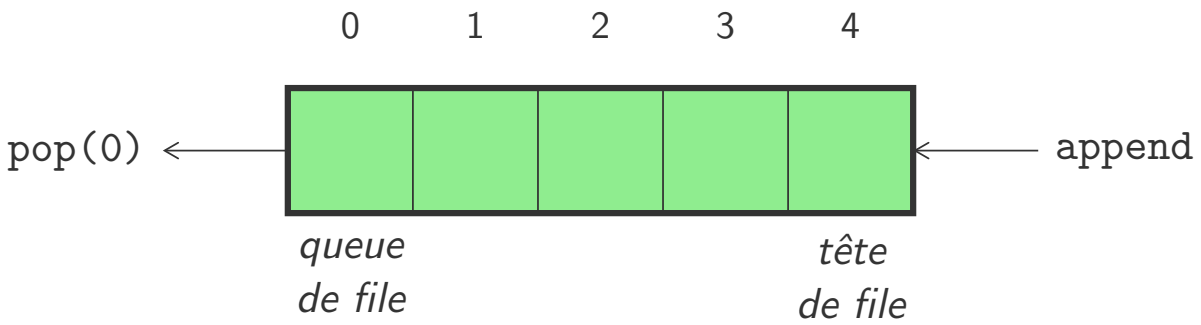
  file d'étudiants aux Ad Hoc par exemple.

- Récupération facile au début de la file.

- Ajout facile à la fin de la file.

- Suit le principe FIFO

  First-In First-Out, premier élément ajouté sera premier à sortir



Files

```
1  queue = []                    # la file est vide
2
3  queue.append(1)               # la file contient [1]
4  queue.append(2)               # la file contient [1, 2]
5  queue.append(3)               # la file contient [1, 2, 3]
6
7  result = queue.pop(0)         # la file contient [2, 3]
8
9  print(result)
10 print(queue)
```

```
3
[2, 3]
```

Autres séquences

Intervalles

- LEs intervalles sont des séquences non modifiables
- Intervalle d'entiers délimités par deux bornes

  Création avec la fonction `range`

```
1  s = "Hello"
2  print(s[1:4])                # ell
3
4  i = range(5, 10)             # de 5 à 10 (non inclus)
5  print(len(i))                # 5
6  print(i[2])                  # 7
```

Parcours d'une liste

- Parcours avec une boucle `while` et un compteur

  Faire varier une variable de 0 à la taille de la liste moins un

- Parcours avec une boucle `for` et l'opérateur `in`

```
1  # Avec une boucle while
2  i = 0
3  while i < len(numbers):
4      print(numbers[i])
5      i += 1
6
7  # Avec une boucle for
8  for n in numbers:
9      print(n)
```

Définition par compréhension

- La boucle `for` permet d'itérer sur les éléments d'une séquence

  Définition par compréhension de séquences

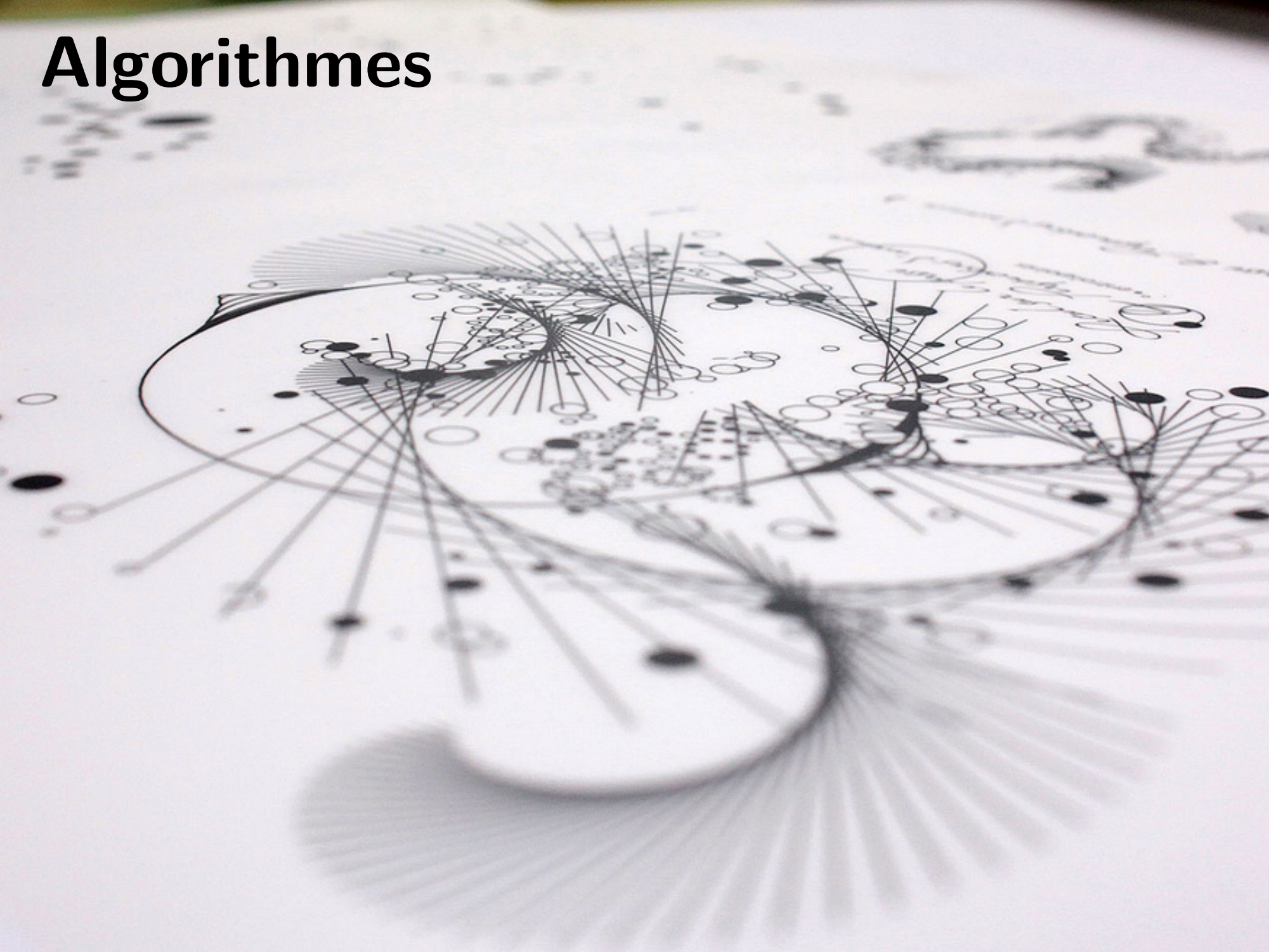
```
1  # Boucle while
2  squares = []
3  i = 0
4  while i <= 100:
5      squares.append(i ** 2)
6      i += 1
7
8  # Boucle for
9  squares = []
10 for i in range(101):
11     squares.append(i ** 2)
12
13 # Définition par compréhension
14 squares = [i ** 2 for i in range(101)]
```

Définition par compréhension

- On peut combiner avec un `if`

```
1  # Définition par compréhension
2  squares = [i ** 2 for i in range(101) if i % 3 == 0]
3
4  # Boucle for correspondant
5  squares = []
6  for i in range(101):
7      if i % 3 == 0:
8          squares.append(i ** 2)
```

Algorithmes



Algorithme

- Un algorithme est une suite d'opérations permettant de résoudre un problème.
- les algorithmes sont indépendants des langages.

  On utilise un pseudo-code pour les décrire
- Exemple de problème :

  recherche de la valeur maximale d'une liste

```
1  Input  : L, est une liste de taille n
2  Output : la valeur maximale de L est renvoyée
3
4  maximum ← L[0]
5  for (i ← 1 to n − 1)
6  {
7      if (L[i] > maximum)
8      {
9          maximum ← L[i]
10     }
11 }
12 return maximum
```

Implémentation de l'algorithme

- Une implémentation d'un algorithme est une réalisation de celui-ci dans un langage particulier.

```
1  def max(liste):
2      maximum = liste[0]
3      for i in range(1, len(l)):
4          if liste[i] > maximum:
5              maximum = liste[i]
6      return maximum
```

Spécification

- Documentation formelle de fonctions avec deux éléments
  - Préconditions sur les paramètres
  - Postconditions sur la valeur de retour
- Les préconditions sont à satisfaire avant l'appel

  Conditions sur les paramètres ou l'état global du programme
- Les postconditions seront garanties après l'appel

  Pour autant que les préconditions étaient satisfaites avant l'appel

Spécification

- Cette documentation peut-être écrite dans un commentaire au dessus de l'implémentation

```
1  # Recherche du maximum d'une liste
2  #
3  # Pre: "liste" est une liste de valeur
4  # Post: La valeur renvoyée est la valeur maximale de "liste"
5  def max(liste):
6      maximum = liste[0]
7      for i in range(1, len(l)):
8          if liste[i] > maximum:
9              maximum = liste[i]
10     return maximum
```

Somme des valeurs d'une liste

```
1  Input  : L, est une liste de taille n
2  Output : la somme des valeurs de L est renvoyée
3
4  sum ← 0
5
6  for (i ← 0 to n − 1)
7  {
8      sum ← sum + L[i]
9  }
10 return sum
```

Somme des valeurs d'une liste (récursif)

```
1  Input  : L, est une liste de taille n,
2  Output : la somme des valeurs de L est renvoyée
3
4  function sum(L, n)
5  {
6      if (n = 0) return 0
7      return L[n − 1] + sum(L, n − 1)
8  }
```

Compter les voyelles d'un string

```
1  Input  : s, est une chaine de caractères de taille n,
2  Output : le nombre de voyelles dans s est renvoyé
3
4  count ← 0
5  for (i ← 0 to n − 1)
6  {
7      if (s[i] est une voyelle)
8      {
9          count ← count + 1
10     }
11 }
12 return count
```

Recherche d'un string

```
1  Input  : s, est une chaine de caractères de taille n,
2           pattern est la chaine de taille m à rechercher dans s
3  Output : l'indice du premier caractère de la première
4           occurence de pattern dans s est renvoyé. None est
5           renvoyé s'il n'y a aucune occurence
6
7  for (i ← 0 to n − m)
8  {
9      j ← 0
10     while (j < m and s[i + j] = pattern[j])
11     {
12         j ← j + 1
13     }
14     if (j = m) return i
15 }
16 return None
```

Les tris

- Le tri des valeurs d'une liste est un exemple de problème pour lequel il existe plusieurs algorithmes :
  - Tri par insertion
  - Tri par sélection
  - Tri à bulles

Tri par insertion

```
1  Input  : L, est une liste de taille n
2  Output : les valeurs de L sont triées dans l'ordre croisant
3
4  for (i ← 1 to n − 1)
5  {
6      x ← L[i]
7      j ← i
8
9      while (j > 0 and L[j − 1] > x)
10     {
11         L[j] ← L[j − 1]
12         j ← j − 1
13     }
14     L[j] ← x
15 }
```

Tri par selection

```
1  Input  : L, est une liste de taille n
2  Output : les valeurs de L sont triées dans l'ordre croisant
3
4  for (i ← 0 to n − 2)
5  {
6      min ← i
7
8      for (j ← i + 1 to n − 1)
9      {
10         if (L[j] < L[min])
11         {
12             min ← j
13         }
14     }
15     échanger L[i] et L[min]
16 }
```

Tri à bulles

```
1  Input  : L, est une liste de taille n
2  Output : les valeurs de L sont triées dans l'ordre croisant
3
4  for (i ← n to 2)
5  {
6      for (j ← 0 to i − 2)
7      {
8          if (L[j] > L[j + 1])
9          {
10             échanger L[j] et L[j + 1]
11         }
12     }
13 }
```

Tri à bulles (récursif)

```
1  Input  : L, est une liste de taille n
2  Output : les valeurs de L sont triées dans l'ordre croisant
3
4
5  function sort(L, n)
6  {
7      if(n > 1)
8      {
9          for (i ← 0 to n − 2)
10         {
11             if (L[i] > L[i + 1])
12             {
13                 échanger L[i] et L[i + 1]
14             }
15         }
16         sort(L, n − 1)
17     }
18 }
```

Crédits

- https ://www.flickr.com/photos/yonpol/4972678968
- https ://www.flickr.com/photos/jack-davies/5663888548
- https ://www.flickr.com/photos/tisane_01/5964046124