Université Paris-Saclay
M2 Droit de la création et du numérique

# Approche de l'élaboration et du fonctionnement des logiciels

Alain Delaët

développer des logiciels

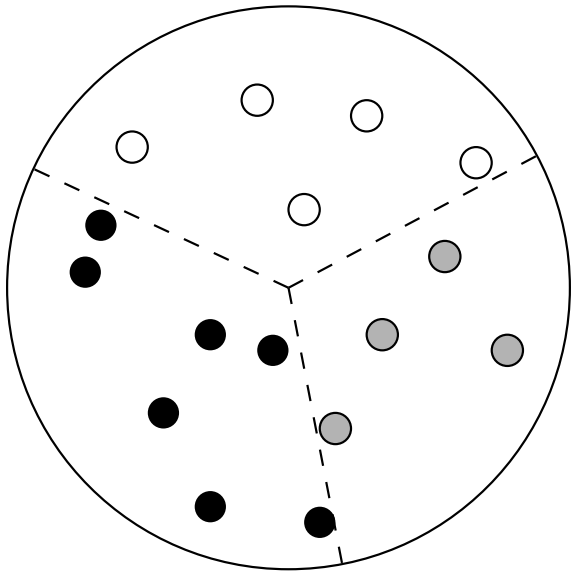
1. intelligence artificielle
2. structures de données
3. développement collaboratif
   - bibliothèque
4. réseaux et Internet
5. le Web
   - HTML et CSS

apporter une réponse **plausible**, mais pas nécessairement exacte, à un problème auquel il est difficile d'appliquer un algorithme traditionnel typiquement,

- donner une réponse exacte prendrait beaucoup trop de temps (exemple : jouer aux échecs)
- pas de définition suffisamment précise du problème (exemple : traduction en français d'une phrase en langue étrangère)
- on part de données incomplètes ou imprécises (exemple : suggérer la meilleure publicité à montrer à un utilisateur)

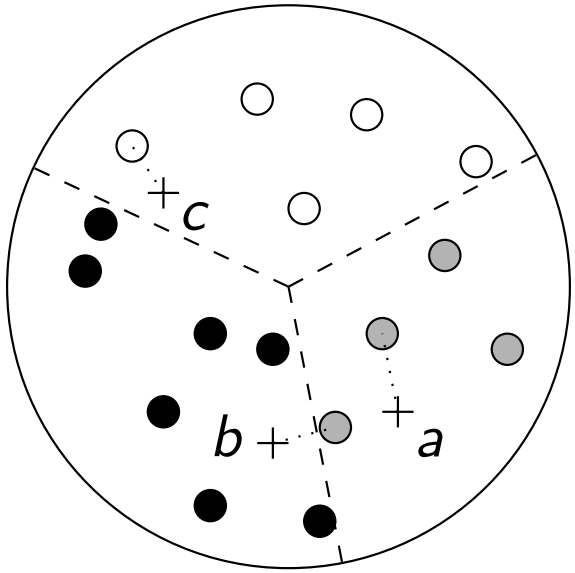
les algorithmes d'**apprentissage** utilisent d'une **grande quantité** d'exemples associant des données d'entrée et les réponses attendues, dont ils se servent pour essayer de **deviner** une réponse convenable sur une nouvelle entrée

des points sont donnés, qui sont coloriés en blanc, gris ou noir



étant donné un nouveau point, on veut deviner sa couleur

on peut regarder la couleur du point le plus proche

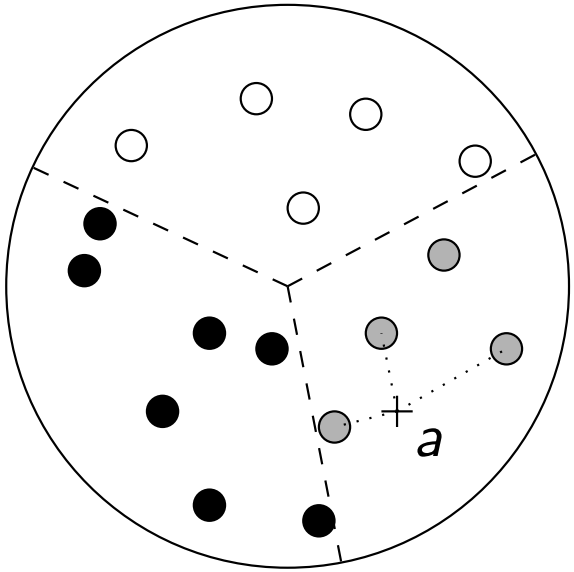



$a$ : ●

$b$ : ●

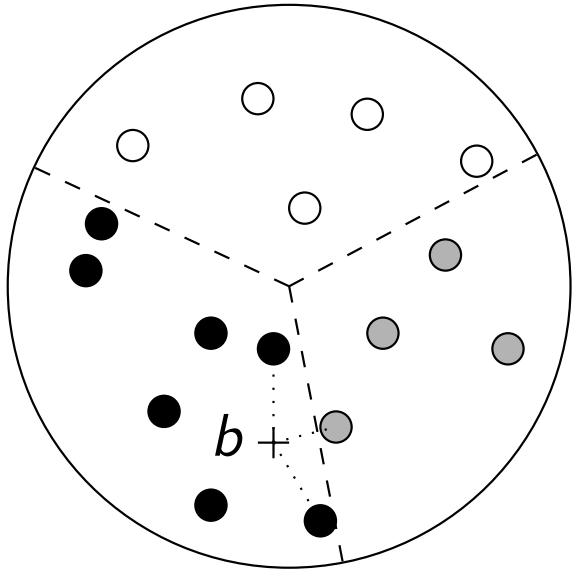
$c$ : ○

on peut regarder la couleur des trois plus proches voisins et retenir la couleur majoritaire



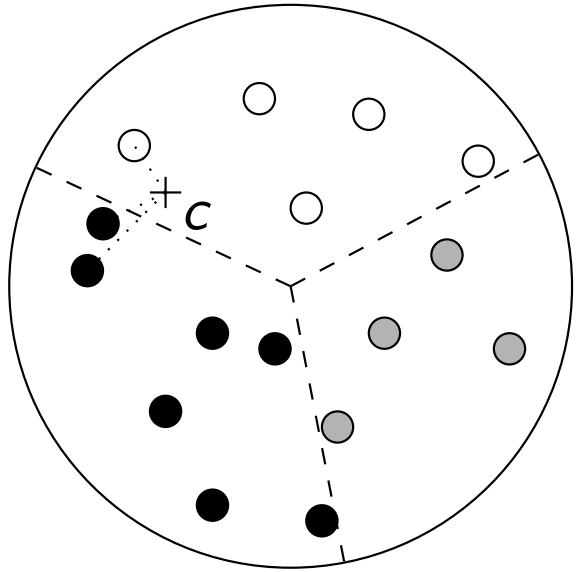

3 voisins : ○○○

résultat : ○




3 voisins : ○●●

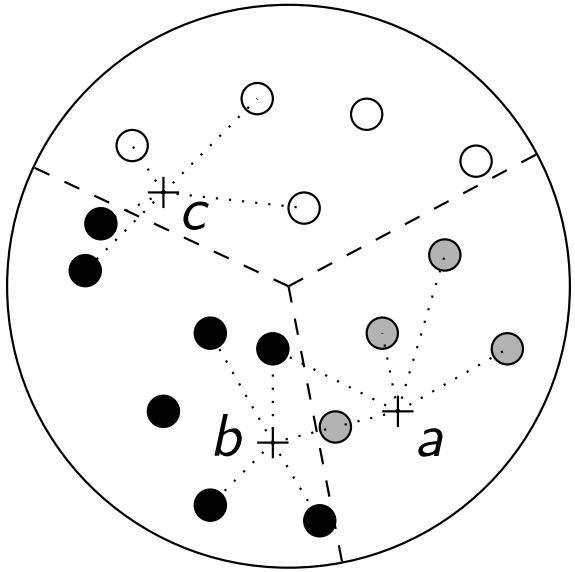
résultat : ●




3 voisins : ○●●

résultat : ●

# cinq plus proches voisins




$a$ : ●●●●●

$b$ : ●●●●●

$c$ : ○●●○○

le résultat est meilleur, mais le calcul coûte plus cher

cet algorithme, dit des $k$ plus proches voisins, peut être utilisé pour une grande variété de notions de ≪ points ≫ et de ≪ distances ≫
exemple :

- les points sont des images
- la distance estime la proximité entre deux images

le traitement automatique des langues par IA (traduction automatique, agent conversationnel, etc.) consiste à **prédire le mot suivant**
l'apprentissage se fait sur un grand nombre de textes, mais aussi sur des requêtes dont on connaît la réponse attendue

## structures de données

une **structure de données** est une façon d'organiser des données en mémoire, de manière à faciliter ensuite des opérations
exemple : une structure d'ensemble permet de

- créer un ensemble vide
- tester si un élément est dans l'ensemble
- ajouter un élément à l'ensemble

il y a une structure d'ensembles fournie par Python
on peut s'en **servir** pour déterminer s'il y a au moins deux valeurs identiques dans un tableau `t` :

```
    vus = set()              # crée un ensemble, initialement vide
    for i in range(n):
    if t[i] in vus:  # l'élément est-il dedans ?
    print(t[i])
    vus.add(t[i])    # ajoute l'élément à l'ensemble
```

mais comment la **réaliser** ?

on peut se servir d'un simple **tableau**

- chercher n'est pas efficace,
  car il faut inspecter tous les éléments

| 17 | 23 | 101 | 42 | 4 | 89 | 13 |
|---|---|---|---|---|---|---|

- ajouter, en revanche, est efficace,
  car Python permet d'ajouter un élément à la fin du tableau

| 17 | 23 | 101 | 42 | 4 | 89 | 13 | **20** |
|---|---|---|---|---|---|---|---|

on peut se servir d'un tableau dans lequel les éléments sont **maintenus triés** par ordre croissant

- chercher est beaucoup plus efficace,
  car on peut faire une **recherche dichotomique**

| 4 | 13 | 17 | 23 | 42 | 89 | 101 |
|---|---|---|---|---|---|---|

- ajouter, en revanche, est moins efficace,
  car il faut insérer l'élément à sa place et donc déplacer des éléments

| 4 | 13 | 17 | **20** | 23 | 42 | 89 | 101 |
|---|---|---|---|---|---|---|---|

pour chercher un élément dans un tableau trié, il existe un algorithme très efficace

1. examiner l'élément central
2. si c'est celui qu'on cherche, c'est terminé
3. s'il est plus grand, chercher à gauche
4. sinon, chercher à droite

on cherche la valeur 22

| 4 | 13 | 17 | 20 | 23 | 42 | 89 | 101 |
|---|---|---|---|---|---|---|---|

on cherche la valeur 22

| 4 | 13 | 17 | 20 | 23 | 42 | 89 | 101 |
|---|---|---|---|---|---|---|---|

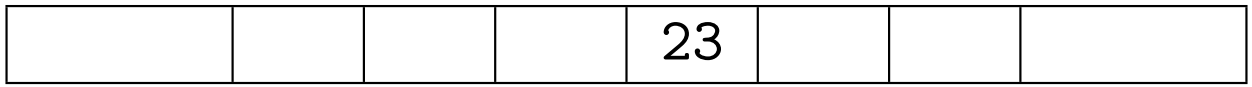
on cherche la valeur 22

| | | | | 23 | 42 | 89 | 101 |
|---|---|---|---|---|---|---|---|

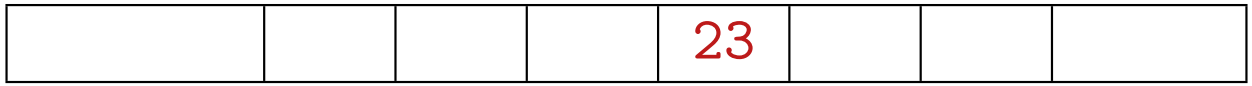
on cherche la valeur 22

| | | | | 23 | 42 | 89 | 101 |
|---|---|---|---|---|---|---|---|

on cherche la valeur 22



on cherche la valeur 22




remarque : on n'a pas eu à considérer toutes les valeurs contenues dans le tableau (mais seulement trois)

# d'autres solutions

il existe plein d'autres solutions pour réaliser la structure de données « ensemble d'entiers », certaines encore plus efficaces (notamment celle cachée derrière le `set()` de Python)

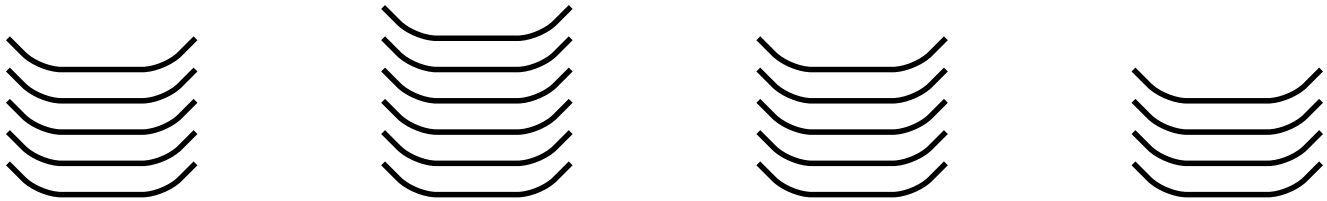
il existe beaucoup d'autres structures de données

- piles
- files
- dictionnaires
- graphes

comme une pile d'assiettes ; on peut

- ajouter sur le dessus
- retirer l'élément au sommet

(en anglais, LIFO, pour *Last In, First Out*)

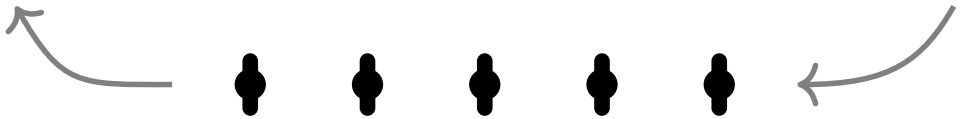


application : l'historique de votre navigateur

comme à la boulangerie ; on peut

- ajouter un élément à la fin
- retirer un élément au début

(en anglais, FIFO, pour *First In, First Out*)



application : des tâches à effectuer

des valeurs sont associées à des clés ; on peut

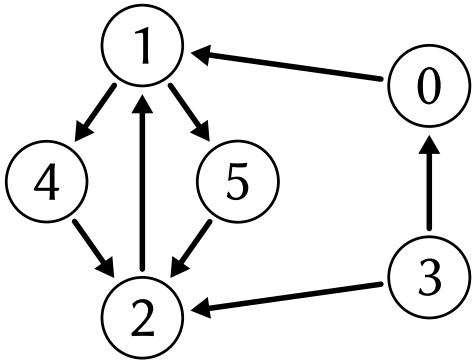
- ajouter une nouvelle entrée
- obtenir la valeur associée à une clé
- supprimer une entrée

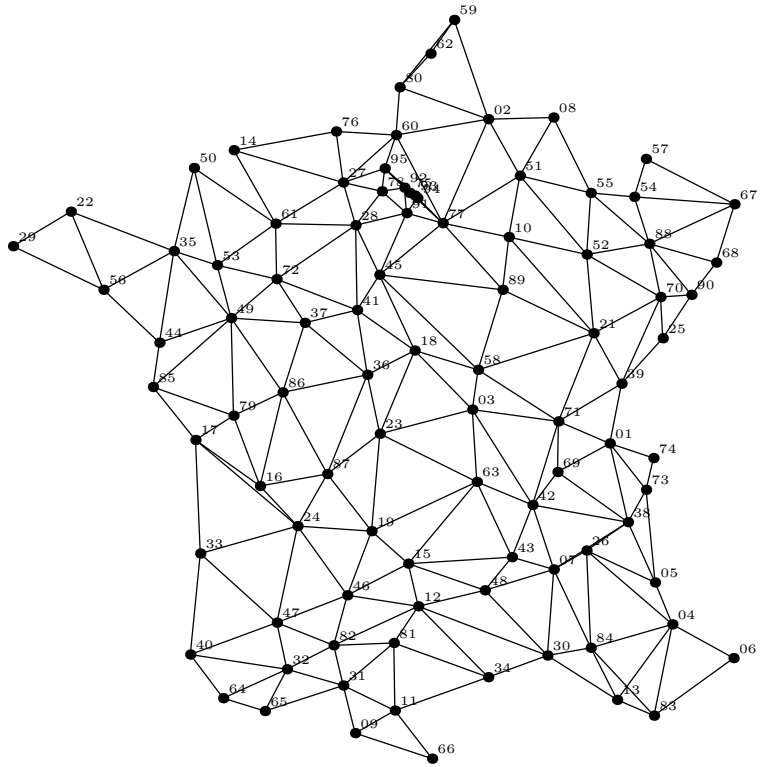
$$\begin{aligned} \text{Alice} &\mapsto 25 \\ \text{Bob} &\mapsto 42 \\ \text{Oscar} &\mapsto 17 \end{aligned}$$

application : le cache de votre navigateur

des sommets sont reliés entre eux par des arcs ; on peut

- ajouter des sommets, des arcs
- obtenir les arcs sortant d'un sommet



applications : réseau routier, réseau informatique, carte, etc.

## développement collaboratif

développer un gros logiciel demande

- de la **collaboration** entre les développeurs
- de la **réutilisation** de (morceaux de) programmes

comment l'organiser ?

on construit des **bibliothèques** de composants logiciels destinés à être réutilisés

en Python, ces composants peuvent être notamment des fonctions, comme la fonction `randint` de la bibliothèque `random`

note : en anglais, on parle de *library* (trop souvent incorrectement traduit en français par *librairie*)

un langage de programmation vient avec sa **bibliothèque standard**
elle est composée de plusieurs bibliothèques élémentaires, chacune avec son usage particulier
ainsi, la bibliothèque standard de Python offre `random`, mais aussi `time`, `sys`, `csv`, etc. (des dizaines au total)

il existe aussi des bibliothèques extérieures à la bibliothèque standard, réalisées par des particuliers ou des entreprises, que l'on peut **installer**
exemple : la bibliothèque `pygame` pour les jeux vidéos
`https://pypi.org/project/pygame/`

# éléments d'une bibliothèque

une bibliothèque vient avec

- des **instructions** d'installation
- une liste d'**auteurs**
- une **licence** logicielle
- une **documentation** relative à son utilisation

une bibliothèque peut être fournie

- sous la forme de code source
- sous la forme de code déjà compilé

l'installation peut se faire

- manuellement, en téléchargeant la bibliothèque depuis un site
- à travers un système de **paquets** du langage de programmation ou du système d'exploitation

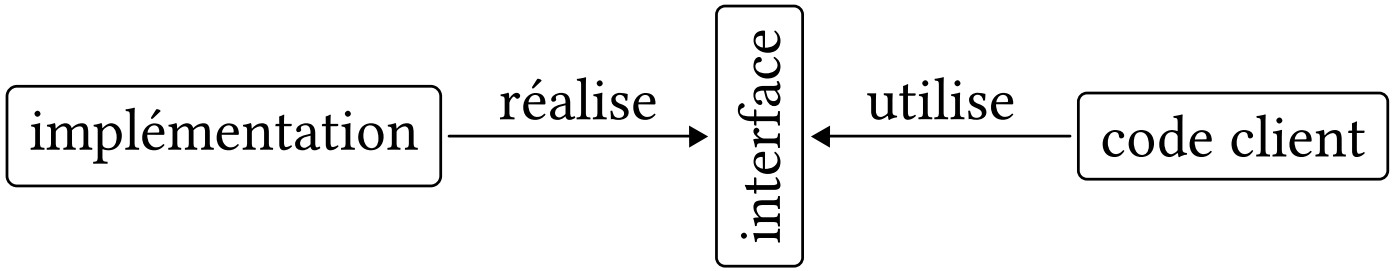
l'installation d'une bibliothèque peut exiger l'installation d'autres bibliothèques, appelées **dépendances**

la **licence** indique les conditions sous lesquelles la bibliothèque peut être utilisée, modifiée ou redistribuée
exemple : la bibliothèque `pygame` est distribuée sous la licence `GNU LGPL version 2.1`, qui permet notamment

- de modifier la bibliothèque, en lui conservant alors la même licence
- d'utiliser la bibliothèque dans un logiciel, quelle que soit sa licence, y compris commerciale

les fonctionnalités offertes par une bibliothèque sont décrites au travers d'une **interface de programmation**, en anglais **API** pour *Application Programming Interface*
c'est la partie **visible** de la bibliothèque
toute une partie du code peut rester cachée, intentionnellement

le code qui utilise la bibliothèque ne voit que l'interface




on appelle cela la **barrière d'abstraction**
exemple : les ensembles de Python utilisés plus haut

en particulier, les parties non visibles de la bibliothèque peuvent être modifiées sans affecter le code client
exemple : demain le générateur pseudo-aléatoire de Python est amélioré, mais la fonction `randint` est toujours là, avec toujours les mêmes paramètres et la même documentation

l'interface inclut notamment la **documentation** (cf cours 4)
exemple :

```
help(randint)
Help on method randint in module random:

randint(a, b) method of random.Random instance
Return random integer in range [a, b], including both end
points.
```

et au-delà, des pages web
`https://docs.python.org/3/library/random.html`

au-delà de la seule notion de bibliothèque, les langages de programmation tentent d'apporter des solutions au problème du développement logiciel au travers de **paradigmes**
l'un des plus répandus est celui de **programmation orientée objet**

un **objet** est la donnée

- d'un état interne
- d'opérations pour interagir avec cet état, appelées méthodes

exemple : un personnage dans un jeu vidéo

- état = position, vitalité, score, etc.
- méthodes = se déplacer, marquer des points, etc.

un objet appartient à une **classe**, qui définit les méthodes disponibles

```
p = Player("Link")  # créer un objet de la classe Player
p.move(10, 20)
if ...:
  p.add_score(100)
...
```

dans le programme vu plus haut,

```
vus = set()              # crée un ensemble, initialement vide
for i in range(n):
  if t[i] in vus:  # l'élément est-il dedans ?
    print(t[i])
  vus.add(t[i])    # ajoute l'élément à l'ensemble
```

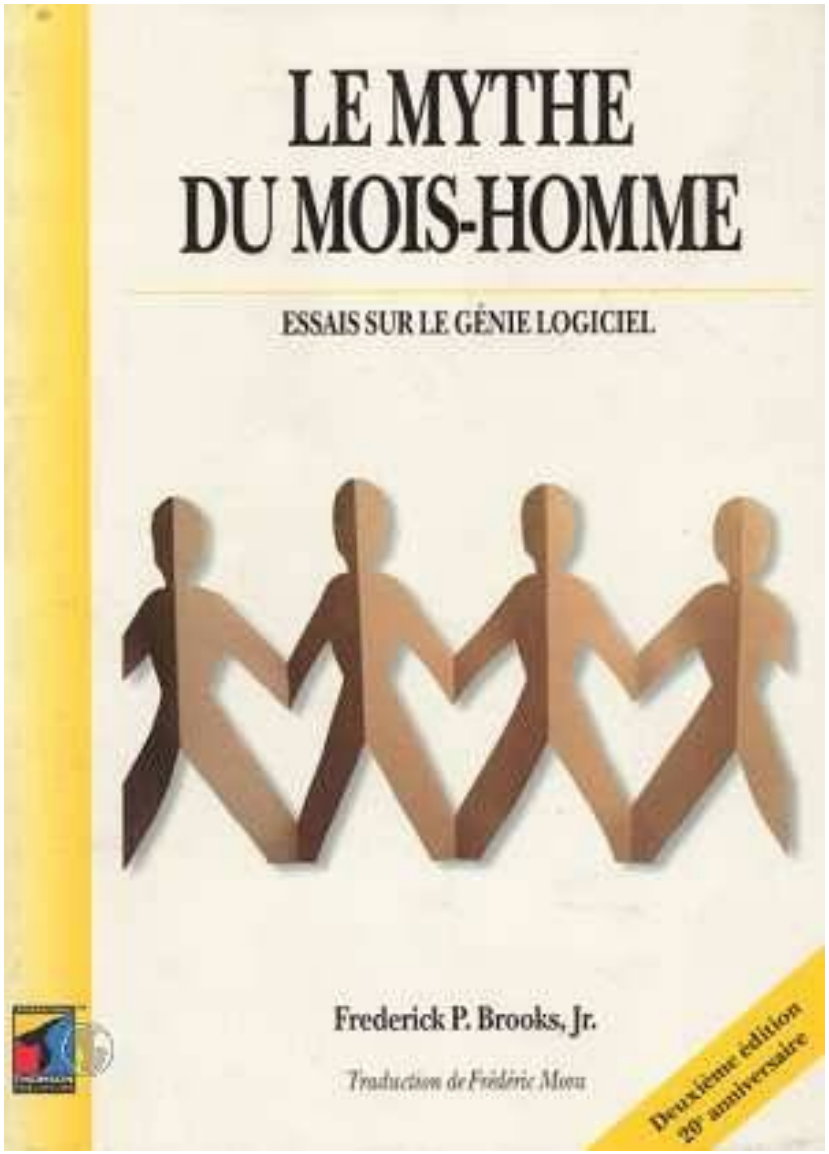
l'ensemble `vus` est un objet (de la classe `set`), qui offre notamment des méthodes pour ajouter un élément et chercher parmi les éléments

il existe bien d'autres paradigmes de programmation (impératif, fonctionnel, logique, événementiel, etc.) et **beaucoup** de langages de programmation (des milliers)
à retenir : il n'y a pas de meilleur langage
développer un gros logiciel est difficile, quel que soit le langage

Louvois, logiciel interarmées de la solde

- projet lancé en 1996
- finalement abandonné en 2013

# un livre célèbre (1975)



pour développer du logiciel à plusieurs, il existe de nombreux outils pour

- la gestion de versions
- le suivi des bugs
- la demande de fonctionnalité
- la gestion des tâches
- la documentation

ces outils sont souvent regroupés dans des services web appelés **forges**

la plateforme GitHub (de l'entreprise GitHub) offre de tels services, gratuitement pour les projets de **logiciels libres** et sinon de façon payante
cf `https://github.com/`
c'est la plus utilisée au monde (65 millions d'inscrits)

- un **algorithme** est une suite d'instructions élémentaires pour résoudre un problème
- une **structure de données** est une façon d'organiser des données en mémoire, pour faciliter les opérations
- une **bibliothèque** regroupe des composants logiciels destinés à être réutilisés

# réseaux et Internet

un **réseau informatique** est un ensemble de **nœuds** (des équipements informatiques) reliés entre eux par des **liens** (câbles de cuivres, fibre optique, liaisons satellites, ondes radios, etc.)
une **interface** est un point de raccordement entre un lien et un nœud (exemple : une carte réseau)

un **protocole** est un ensemble de règles permettant d'établir, mener et terminer une communication entre deux entités
un protocole peut notamment garantir

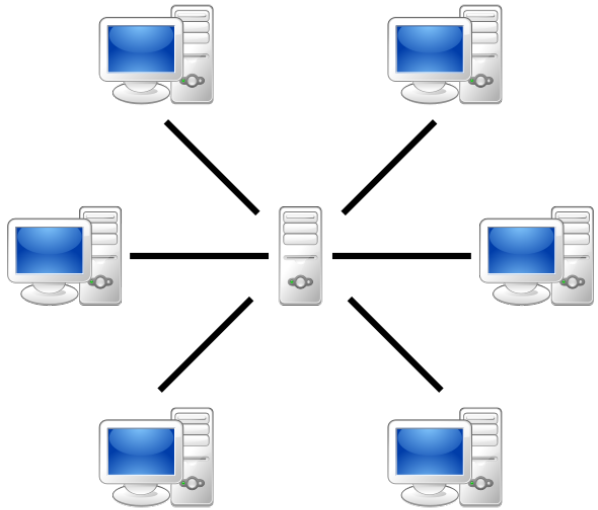
- l'absence d'erreur
- l'efficacité
- la confidentialité

exemple : le protocole HTTP
c'est Internet Engineering Task Force (**IETF**) qui s'occupe de maintenir les protocoles

un **serveur** désigne à la fois la machine et le logiciel exécutant un service
exemple : le serveur Web Apache est un logiciel offrant le service de distribution de pages Web via le protocole HTTP
un **client** désigne à la fois la machine et le logiciel se connectant par le réseau à un serveur
exemple : un navigateur Web

dans le modèle **client-serveur**,

- un serveur est en attente de connexions
- les clients se connectent et sont traités de manière individuelle
- les clients ne communiquent pas entre eux



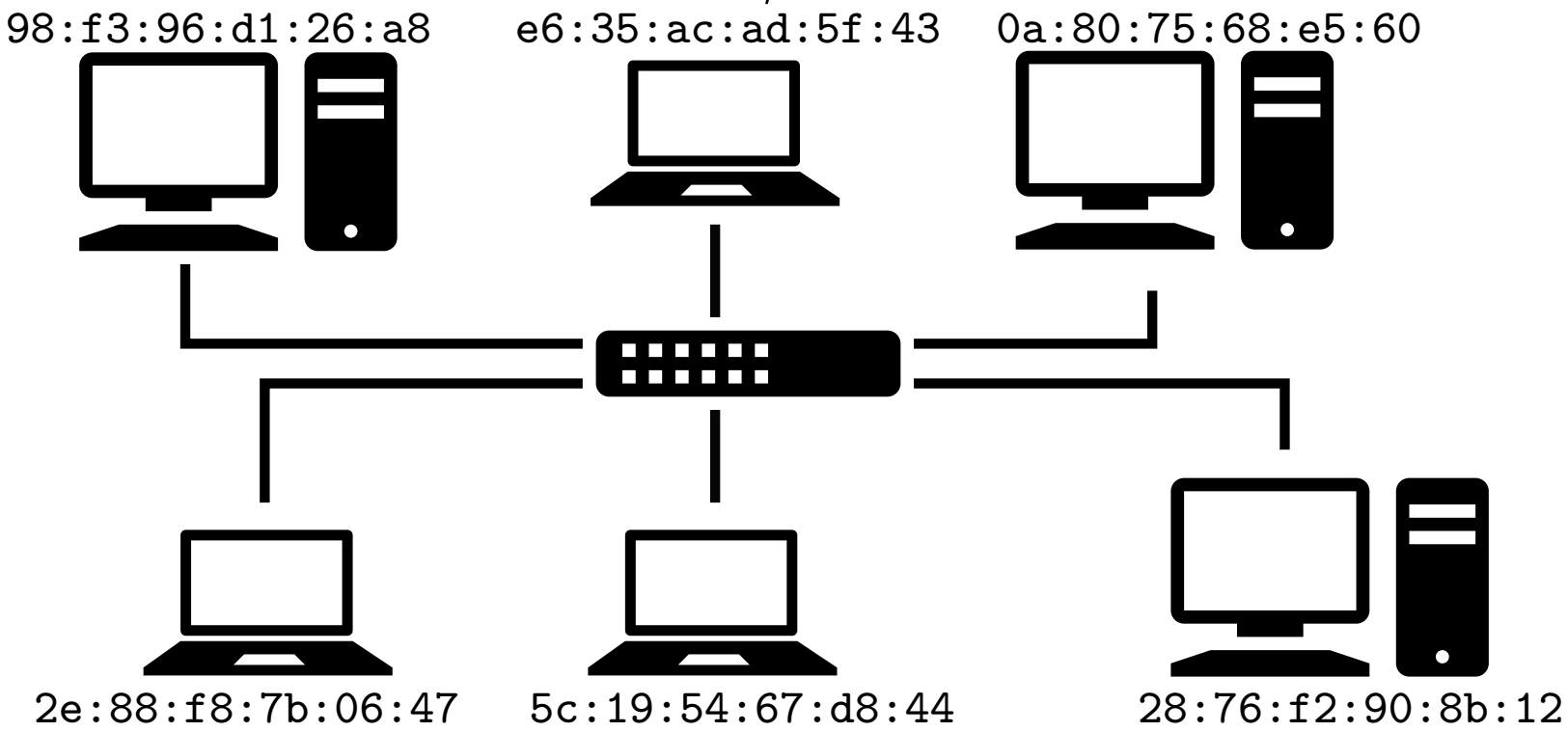
on parle d'architecture **en étoile**

un réseau informatique global conçu en 1970
fonctionne grâce à un ensemble de protocoles, organisés en quatre couches

| nº | nom | exemple de protocole |
|---|---|---|
| 4 | application | HTTP, POP, IMAP |
| 3 | transport | TCP, UDP |
| 2 | réseau | IPv4, IPv6, ICMP |
| 1 | liaison | Ethernet, Wi-Fi |

pour des machines reliées **directement** entre elles (câbles Ethernet ou réseau Wi-Fi), on parle de réseau local ou **LAN** (*Local Area Network*)
chaque interface matérielle (carte réseau) possède une adresse sur 6 octets dite **adresse MAC** (exemple : `98:f3:96:d1:26:a8`)
les machines sont reliées entre elles par un périphérique réseau appelé **commutateur** (en anglais **switch**) faisant office de ≪ multi-prise ≫

six machines connectées à un *switch*, et leurs adresses MAC



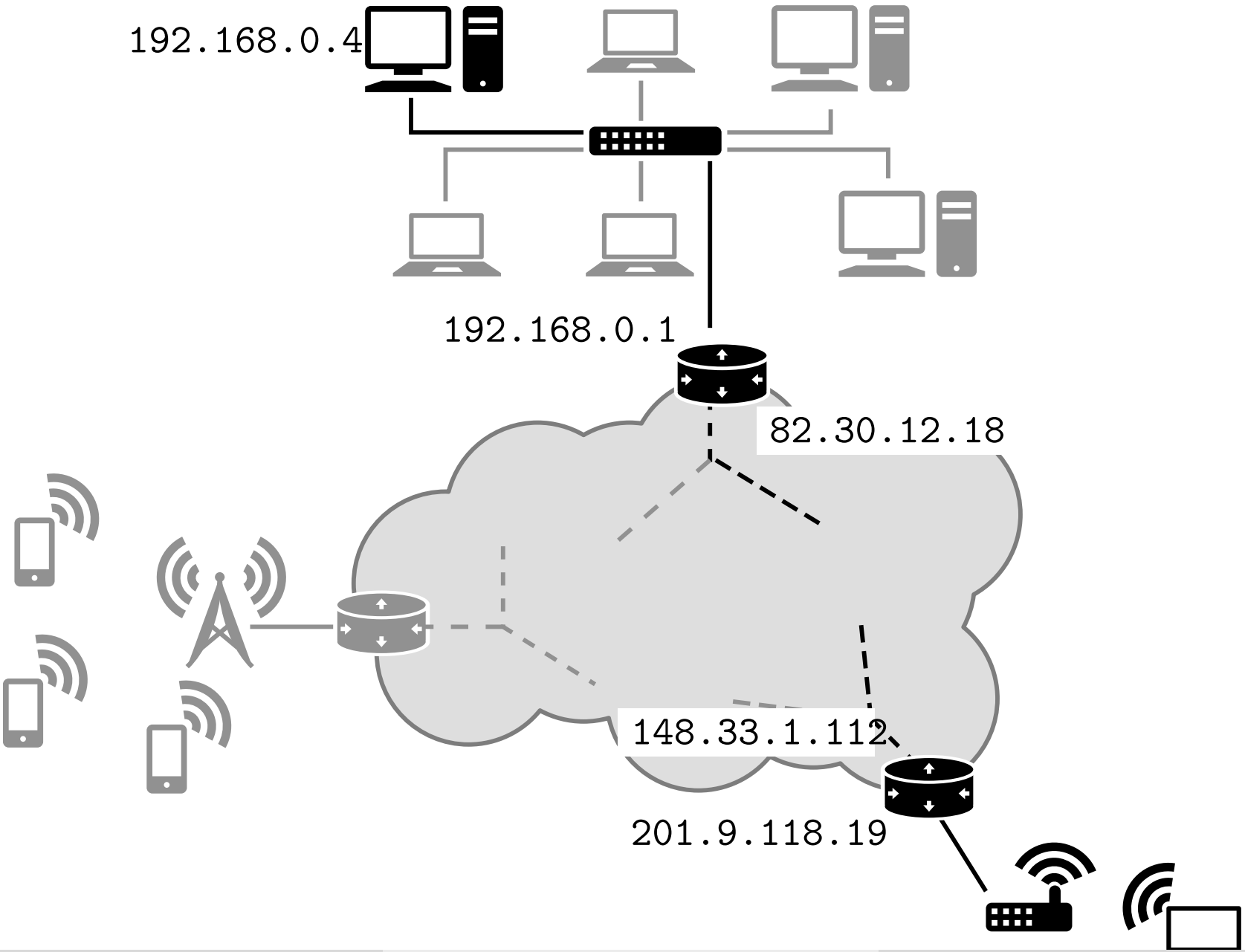
lorsqu'une machine souhaite communiquer avec une autre machine, elle envoie sur son câble un paquet d'octets appelé **trame Ethernet**


8 octets | 6 octets | 6 octets | 2 octets | *longueur* octets | 16 octets

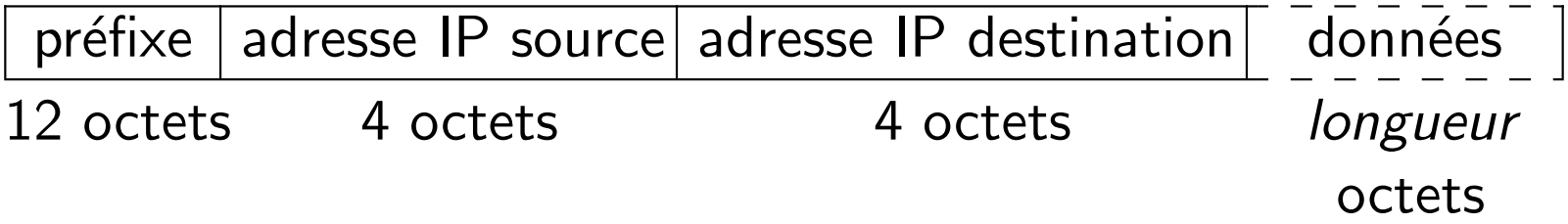
le switch décode l'adresse de destination et envoie la trame sur le câble correspondant

on veut relier ensemble différents réseaux locaux pour permettre à deux machines de communiquer
pour cela, le **protocole IP** (pour *Internet Protocol*)

- associe à (chaque interface de) chaque machine un identifiant unique appelé **adresse IP** (par exemple `192.168.0.4`)
- définit la notion de **sous-réseau** (machines toutes connectées entre elles) et de **passerelle** (en anglais *gateway*)


192.168.0.4
192.168.0.1
82.30.12.18
148.33.1.112
201.9.118.19

lorsqu'une machine souhaite envoyer des données à une autre machine, elle les encapsule dans un **paquet IP**




ce paquet est lui-même encapsulé dans une trame Ethernet

- si le paquet est à destination d'une machine du même sous-réseau, il est envoyé directement
- sinon, il est envoyé à la passerelle, qui
  - extrait le paquet IP de la trame Ethernet
  - le ré-encapsule dans une nouvelle trame
  - le transmet sur une autre interface

les passerelles possèdent des **tables de routage** (adresses joignables et interfaces à utiliser)

on sait maintenant envoyer des données d'une *machine* à une autre *machine*, mais

- une même machine peut offrir plusieurs services (mail, Web, etc.)
- des paquets peuvent se perdre en cours de route

c'est le rôle du protocole **TCP** d'y remédier

le protocole TCP (*Transmission Control Protocol*) ajoute

- un **numéro de port** identifiant un service sur une machine
  exemple : le port 80 pour un serveur HTTP
- un système d'**accusés de réception** pour garantir le bon acheminement des données
  - l'expéditeur découpe les données en paquets numérotés
  - si le destinataire les reçoit dans le désordre, il peut les réordonner
  - si le destinataire ne reçoit pas certains paquets, il peut les redemander
  - si le destinataire reçoit des paquets en double, il peut les ignorer

le protocole TCP utilise ses propres paquets,
eux-mêmes contenus dans des paquets IP
eux-mêmes contenus dans des paquets Ethernet !

le **système de résolution de noms** ou **DNS** (pour *Domain Name System*) fait correspondre des noms symboliques à des adresses IP
exemple :

| nom | adresse IP |
|---|---|
| `universite-paris-saclay.fr` | `129.175.212.146` |
| `wikipedia.fr` | `141.94.212.184` |
| `webmail.free.fr` | `212.27.48.1` |
| ... | ... |

c'est lui-même un service, avec son protocole et ses serveurs

# le Web

le **Web** (abréviation de l'anglais **World Wide Web**), ce sont des documents reliés entre eux par des **liens hypertexte**
par extension, le Web désigne

- les langages informatique permettant d'écrire les documents hypertextes, à savoir **HTML** et **CSS**
- une architecture client-serveur utilisant le protocole **HTTP** (*Hypertext Transfer Protocol*) ou sa version sécurisée **HTTPS** (*HTTP Secure*)
- des langages de programmation côté serveur, comme PHP
- des langages de programmation côté client, comme JavaScript

| | |
|---|---|
| 1980–1984 | création du langage de balisage SGML (*Standard Generalized Markup Language*) |
| 1986 | SGML devient la norme ISO 8879:1986. |
| 1991 | Tim Berners-Lee annonce la première version du Web |
| 1993 | les premiers navigateurs Web graphiques apparaissent |
| 1994–1997 | création du **World Wide Web Consortium** (W3C), organisme de normalisation |
| 2000–2007 | le Web se développe de manière rapide mais anarchique |
| 2007–2014 | long effort de modernisation et de standardisation du standard HTML pour donner HTML 5 en 2014 |

le langage **HTML** (pour l'anglais *HyperText Markup Language* ou langage de balisage hypertexte) est un format textuel permettant de décrire le contenu et la structure d'un document


```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <title>Le titre de la première page</title>
    <meta charset="utf-8" />
  </head>
  <body>
    <h1>Première page</h1>
    <p>Notre toute première page !</p>
    <!-- le reste du document peut venir ici -->
  </body>
</html>
```

le fichier possède une **structure**, définie par des **balises** HTML

- des balises ouvrantes, comme `<head>`
- des balises fermantes, comme `</head>`
- des balises vides, comme `<meta charset="utf-8" />`

une balise comme `<meta charset="utf-8" />` définit l'**attribut** `charset` en lui donnant la valeur `"utf-8"`
de même, la balise `<html lang="fr">` définit l'attribut `lang`

une paire de balises ouvrante et fermante, ainsi que le contenu situé entre les deux, est appelé un **élément**

exemple : `<h1>Première page</h1>`

les caractères délimités par `<!--` et `-->` forment un commentaire

le format HTML est standardisé par le W3C (des centaines de pages)
un validateur en ligne permet de vérifier si un fichier HTML est conforme au standard

`https://validator.w3.org`

un unique élément `html`, avec un attribut `lang`, contenant uniquement deux sous-éléments, `head` et `body`

```
<!DOCTYPE html>
<html lang="fr">
 <head>
   <title> Un titre </title>
   <meta charset="utf-8" />
 </head>
 <body>
   <!-- Le contenu du document -->
 </body>
</html>
```

les balises doivent être **bien parenthésées**, c'est-à-dire qu'une balise ouvrante est fermée par une balise correspondante, au même niveau
exemples :

- la séquence `<b> ... <i> ... </i> ... </b>` est valide
- la séquence `<b> ... <i> ... </b> ... </i>` est invalide

```
<h1>Un titre de section</h1>
<h2>Premier sous-titre</h2>
<h2>Deuxième sous-titre</h2>
<h3>Et encore</h3>
<h4>Et encore</h4>
<h5>Et encore</h5>
<h6>Dernier niveau</h6>
<h1>Section suivante</h1>
```

# Un titre de section

## Premier sous-titre

## Deuxième sous-titre

### Et encore

#### Et encore

##### Et encore

###### Dernier niveau

# Section suivante

```
<p>Un premier paragraphe. On
  y met du texte
n'importe comment et il est
affiché comme il   faut.</p>
<p>Si on commence un nouveau
paragraphe, on remarque
un retour à la ligne.</p>
```

Un premier paragraphe. On y met du texte n'importe comment et il est affiché comme il faut.

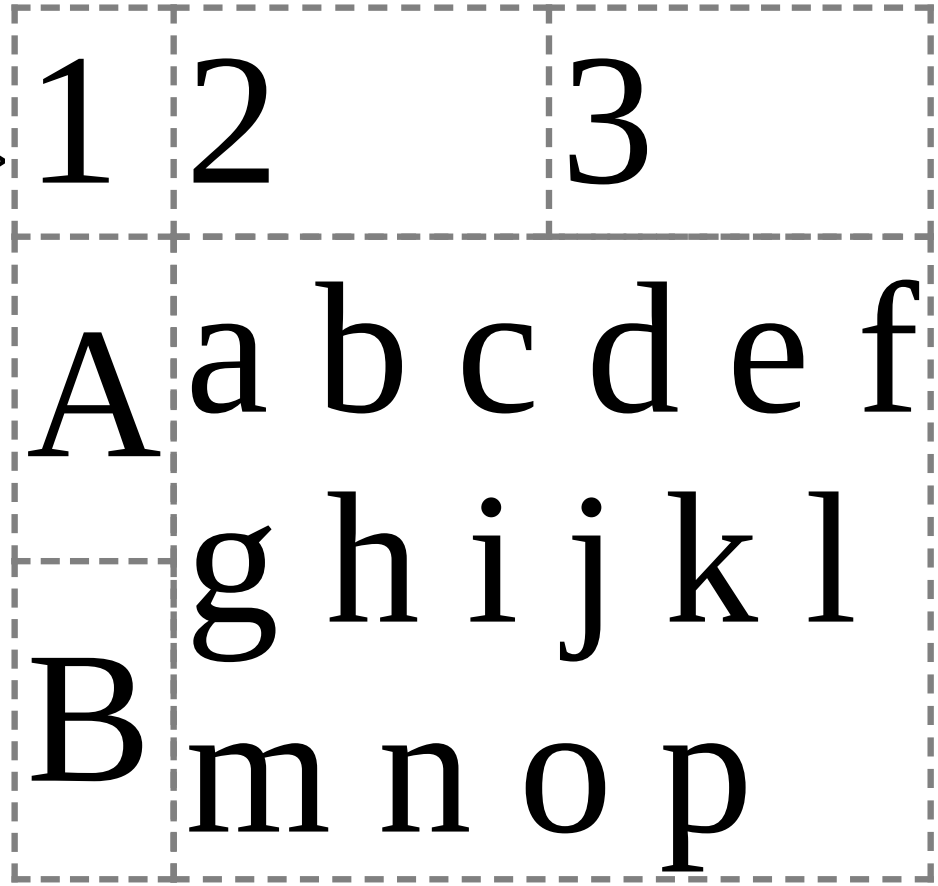
Si on commence un nouveau paragraphe, on remarque un retour à la ligne.

```
<ul>
 <li>Une chose</li>
 <li>Puis une autre</li>
</ul>
<ol>
 <li>Le premier point</li>
 <li>Le second point</li>
 <li>
  <ol>
    <li>Un sous-point du
        troisième point</li>
    <li>Autre chose</li>
  </ol>
 </li>
</ol>
```

- Une chose
- Puis une autre

1. Le premier point
2. Le second point
3. 
    1. Un sous-point du troisième point
    2. Autre chose

```
<table>
  <tr>
  <td>1</td><td>2</td><td>3</td>
  </tr>
  <tr>
    <td>A</td>
    <td rowspan="2"
        colspan="2">
      a b c d e f g h i
      j k l m n o p
    </td>
  </tr>
  <tr> <td>B</td> </tr>
</table>
```



```
Mettre du texte en <b>gras</b>,
<i>italique</i>, <u>souligné</u>,
<mark>surligné</mark>,
<small>petit</small>,
<code>code source</code>. On peut
aussi mettre le texte en indice
comme dans 1000101<sub>2</sub>
ou en exposant comme dans
2<sup>3</sup>. Les balises peuvent
être <i><b><u>combinées</u></b></i>.
```

Mettre du texte en **gras**, *italique*, souligné, surligné, petit, `code source`. On peut aussi mettre le texte en indice comme dans $1000101_2$ ou en exposant comme dans $2^3$. Les balises peuvent être ***combinées***.

vers un autre fichier HTML

```
<a href="fichier.html">Un lien <b>hypertexte</b></a>
```

Un lien **hypertexte**

ou encore vers une page extérieure

```
<a href="www.ici.fr/toto.html">Cliquez ici !</a>
```

```
<img src="html_logo.svg" alt="Logo" />
```

les navigateurs supportent de nombreux formats

- `jpeg` : compressé avec perte
- `png` : compressé sans perte
- `gif` : 256 couleurs seulement, mais avec animations
- `svg` : images vectorielles

le langage **CSS** (pour *Cascading Style Sheets* ou feuilles de style en cascade) permet de définir les propriétés graphiques des éléments HTML constituant une page Web

```
<p style="text-align:right;">
On écrit un
<span style="border:1pt solid black;">
paragraphe  </span> de texte, mais
cette fois on <span style="color:gray">
modifie </span> le style graphique
<span style="border:1pt solid black;">
des éléments. </span>
</p>
```

On écrit un paragraphe de texte, mais cette fois on modifie le style graphique des éléments.

- l'attribut `style` ajoute des **propriétés CSS**
- la balise `<span>` est une balise neutre

il y a plusieurs inconvénients à cette approche :

- l'attribut `style` rend le code HTML peu lisible
- l'attribut `style` est **dupliqué** $\Rightarrow$ maintenance compliquée

d'où l'idée de **séparer** la structure du document (HTML) d'une part et sa présentation graphique (CSS) d'autre part

```
<!DOCTYPE html>
<html lang="fr">
 <head>
  <title>CSS 2</title>
  <meta charset="utf-8" />
  <style>
    p { text-align: right; }
    .bord { border:1pt solid black; }
    #n42 { color: gray; }
  </style>
 </head>
 <body>
  <p>
   On écrit un <span class="bord">paragraphe</span>
   de texte, mais cette fois on <span id="n42">modifie</span>
   le style graphique <span class="bord">des éléments.</span>
  </p>
 </body>
</html>
```

les informations de style

- ont disparu de la partie HTML, remplacées par des attributs `class` et `id`
- sont déportées dans une balise `<style>` présente dans l'entête du document, qui contient des **sélecteurs CSS**

  `p { text-align: right; }`

  toutes les balises `<p>` du document sont justifiées à droite

  `.bord { border: 1pt solid black; }`

  les balises possédant la valeur `"bord"` dans leur attribut `class` ont une bordure

  `#n42 { color: gray; }`

  l'unique balise dont l'attribut `id` vaut `"n42"` doit avoir un texte de couleur grise

# quelques propriétés CSS (1/2)

| propriété | valeur | description |
|---|---|---|
| `display` | `none`, `block` ou `inline` | mode d'affichage de la boîte |
| `background` | couleur | couleur de fond de la boîte |
| `color` | couleur | couleur de texte |
| `border` | taille motif couleur ou `none` | le motif est `solid`, `dotted` ou `dashed` |
| `margin` | longueur | taille des marges |
| `padding` | longueur | taille des ajustements |
| `text-decoration` | `none`, `underline`, `overline` ou `line-through` | décoration du texte |

# quelques propriétés CSS (2/2)

| propriété | valeur | description |
|---|---|---|
| `text-align` | `left`, `right`, `center` ou `justify` | justification du texte |
| `font-family` | `fixed`, `serif` ou `sans-serif` | nom de la police |
| `font-weight` | `normal`, `light`, `bold` ou `bolder` | graisse de la police |
| `font-style` | `normal` ou `italic` | style de la police |
| `font-size` | longueur ou `xx-small`, `x-small`, `small`, `normal`, `large`, `x-large` ou `xx-large` | taille de la police |

objectif : réaliser une page web en HTML et CSS, par exemple pour faire son CV

1. télécharger le fichier `page.html` depuis eCampus
2. le téléverser dans JupyterLab
3. faire un clic droit sur le nom du fichier, et choisir `Open With` puis `HTML Viewer`
4. éditer `page.html` dans JupyterLab ; le rendu est mis à jour automatiquement