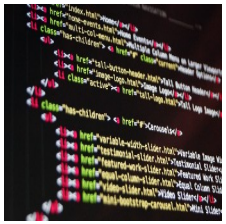
Informatique pour les Sciences de la Vie

Stéphane Téletchéa, maître de conférences en bioinformatique structurale

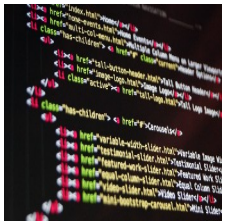
https://ufip.univ-nantes.fr/staff/teletchea-stephane/

Objectifs



- (Re)-Découvrir l'importance de l'**informatique** en **biologie**
- Mettre en place une **stratégie d'analyse** d'un problème de biologie pour y apporter une réponse informatique
- Maîtriser les concepts fondamentaux de la **programmation**
- Comprendre et mettre en œuvre un **langage informatique**
- *Approche différente des années précédentes !*

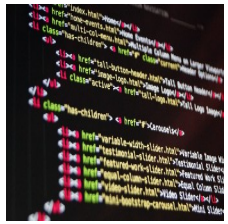
Organisation de l'UE



- 2 Cours magistraux
- 3 TD
- 10 TP
- Évaluations et distanciels d'accompagnement

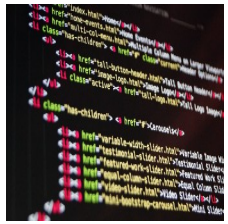
Evaluation :

1) Contrôle continu sur MADOC (30%)
2) Examen de TP (20%)
3) Partiel final (50%)



Intérêt de l’informatique en biologie

Des données complexes et massives



Données massives (***big data*** == Biologie)

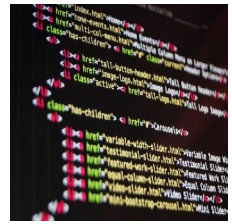
Traitements complexes à enchaîner

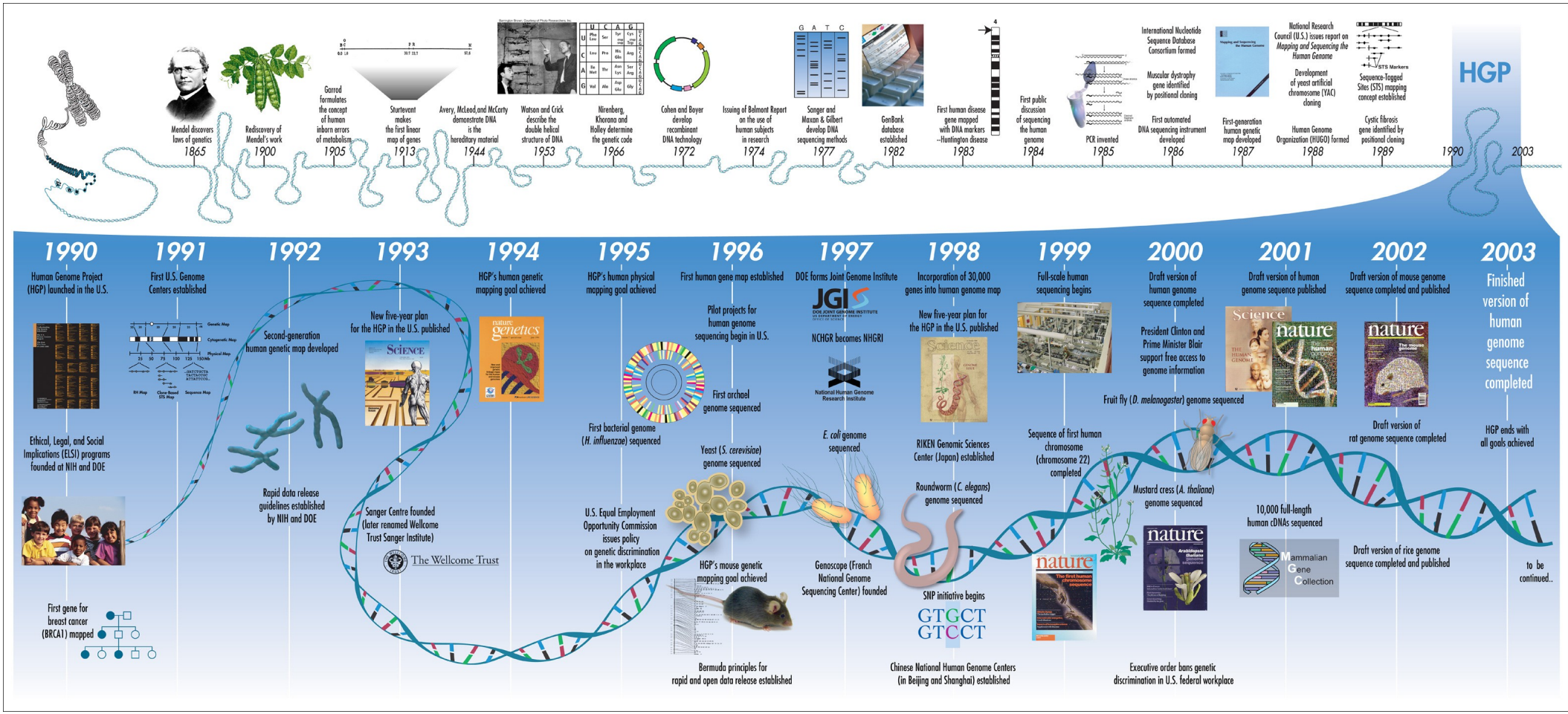
Données de différentes sources, très hétérogènes,avec différents niveaux de précision

Certains problèmes sont impossibles à traiter avec une approche « naïve », ou avec des logiciels « familiers » (tableur, analyse « à la main », etc).

Beaucoup de traitements sont *déjà réalisés, il suffit de savoir les réutiliser*

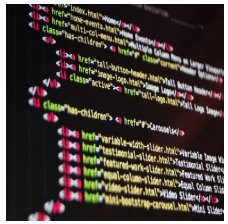
Des données massives issues du séquençage des génomes

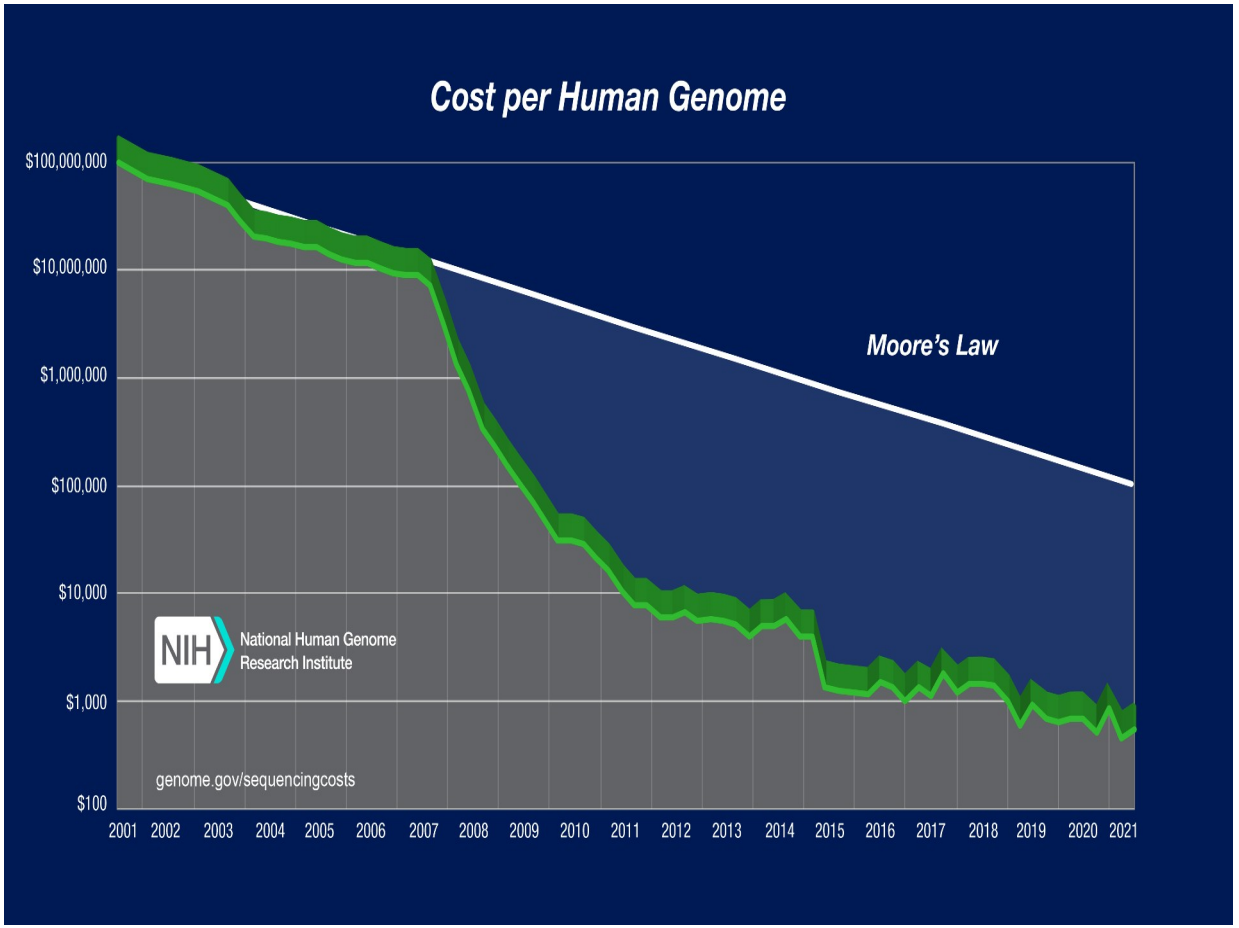




Introduction complète : https://www.nature.com/scitable/ebooks/the-human-genome-project-16553838
Et encore plus (vidéos) : https://www.yourgenome.org/facts/timeline-history-of-genomics

La révolution : le Next Generation Sequencing





1 génome humain :

200 / 500 Go *brut* (NGS)
1 Go compressé
>3 Go décompressé ...

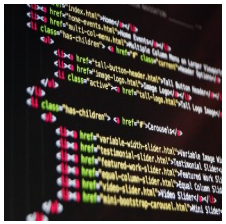
Stockage NGS public : **100 PB**

Facebook : **200 PB**
YouTube : **380 PB**/an

PetaBit : 1000 TB
(~ 1000 disques durs)

Evolution coût par base : https://www.genome.gov/27541954/dna-sequencing-costs-data/
NGS : http://dnasequencing.yolasite.com/next-generation-sequencing.php
Big data en biologie : http://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.1002195
Traitement des données : http://www.strand-ngs.com/support/ngs-data-storage-requirements

De plus en plus de génomes ...



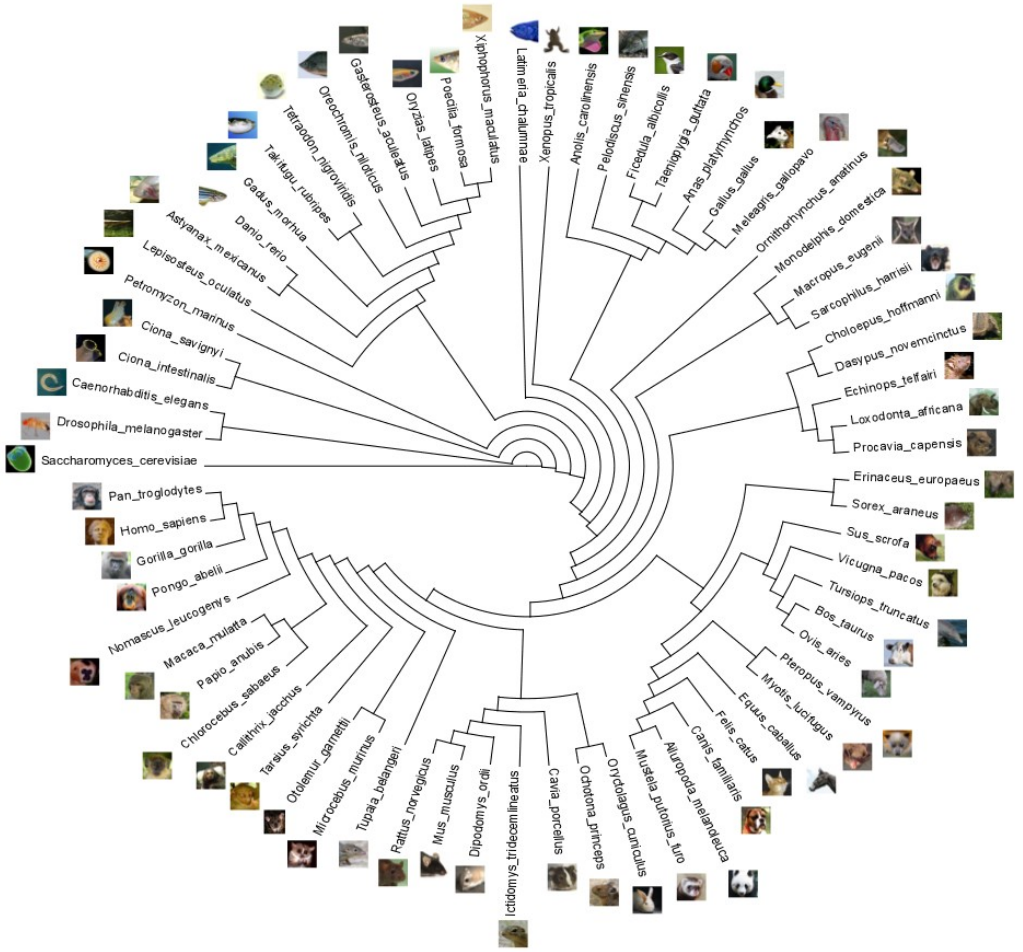


Image obtained using Dendroscope (D.H. Huson and C Scornavacca, Dendroscope 3: An interactive tool for rooted phylogenetic trees and networks, Systematic Biology, 2012)

310 espèces

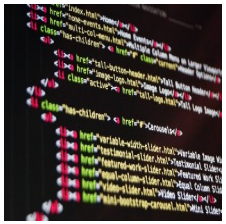
2504 génomes humains
Issus de 26 ethnies
(http://www.internationalgenome.org)

>30 000 genomes en tout …
(bactéries, levures, OTU ...)

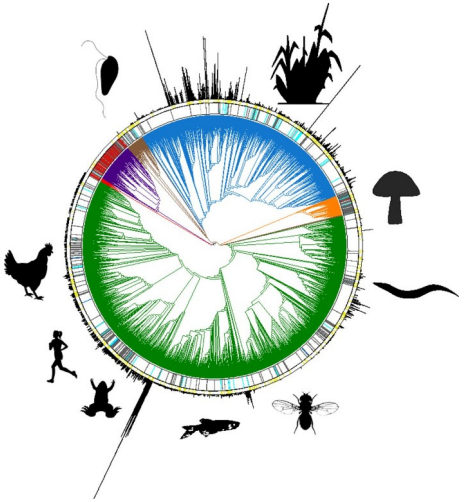
Progression exponentielle

http://www.ensembl.org/info/website/tutorials/index.html

Séquencer tout le vivant

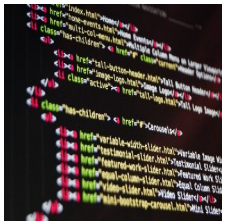


- 1977 : Carl Woese indique qu'il faut recenser tous les ARNs 16S / 18S pour revisiter la classification du vivant
- 2001 : premier génome humain
- 2014 : 2504 génomes humain
- 2015- :
  - 10 000 oiseaux : https://b10k.genomics.cn/
  - 1 000 poissons : https://db.cngb.org/fisht1k/
  - 1 000 plantes : https://db.cngb.org/onekp/
  - 100 diatomées : https://jgi.doe.gov/csp-2021-100-diatom-genomes/
- 2018- : Earth Biogenome Project
  - https://www.earthbiogenome.org/
- 2020-2022 : Beyond 1 Million European
  - https://b1mg-project.eu/



https://fr.wikipedia.org/wiki/Carl_Woese
Earth Biogenome Project : https://www.pnas.org/content/115/17/4325

Chez l'Homme seulement



Statistics

Summary

Assembly	GRCh38.p13 (Genome Reference Consortium Human Build 38), INSDC Assembly GCA_000001405.28, Dec 2013
Base Pairs	3,096,649,726
Golden Path Length	3,096,649,726
Assembly provider	Genome Reference Consortium
Annotation provider	Ensembl
Annotation method	Full genebuild
Genebuild started	Jan 2014
Genebuild released	Jul 2014
Genebuild last updated/patched	Aug 2021
Database version	105.38
Gencode version	GENCODE 39

Gene counts (Primary assembly)

Coding genes	20,465 (incl 653 readthrough)
Non coding genes	24,849
Small non coding genes	4,865
Long non coding genes	17,763 (incl 308 readthrough)
Misc non coding genes	2,221
Pseudogenes	15,217 (incl 6 readthrough)
Gene transcripts	245,000

Gene counts (Alternative sequence)

Coding genes	3,053 (incl 26 readthrough)
Non coding genes	1,555
Small non coding genes	297
Long non coding genes	1,071 (incl 25 readthrough)
Misc non coding genes	187
Pseudogenes	1,799
Gene transcripts	21,638

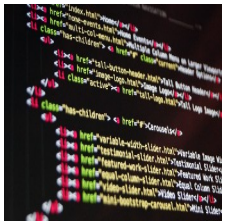
Diminue

Augmente

Patch14 :

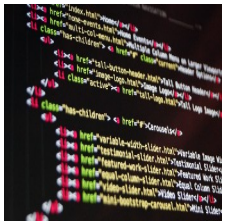
19 830 (protein-)coding genes
26 462 non-coding

http://www.ensembl.org/Homo_sapiens/Info/Annotation (GRChR38.**p14**)



Obtention de ces données biologiques numérisées

Séquençage



Séquençage : exemple de détermination des variants de la protéine Spike, génome pour un cancer une pathologie rare, etc.

NovaSeq 6000, Illumina

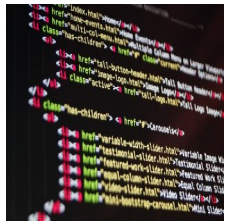
Exemples de traitement pour un génome donné

Temps de préparation de l'échantillon : 6h

Temps de séquençage : ~ 24h

Temps d'analyse : ~ 30 minutes

https://umr1087.univ-nantes.fr/home/genobird-acquires-the-latest-high-throughput-sequencer-generation

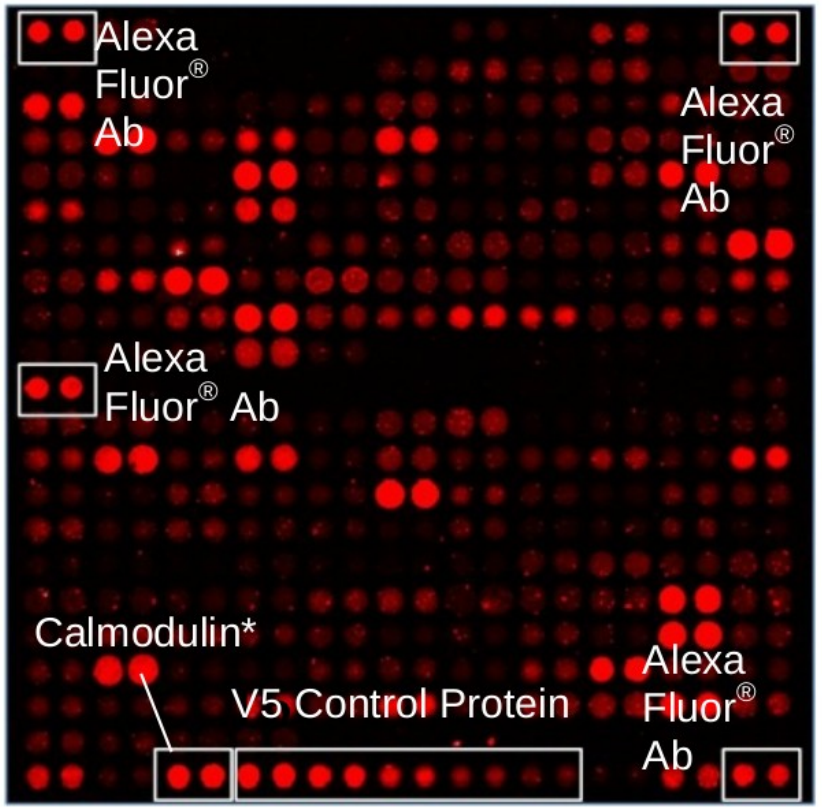


Protéomique

Puces à protéines : plateforme IMPACT

https://plateforme-impact.cnrs.fr/

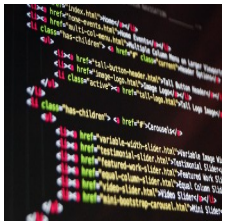
→ Production de puces à protéines, jusqu'à 10 000 analyses par puce !



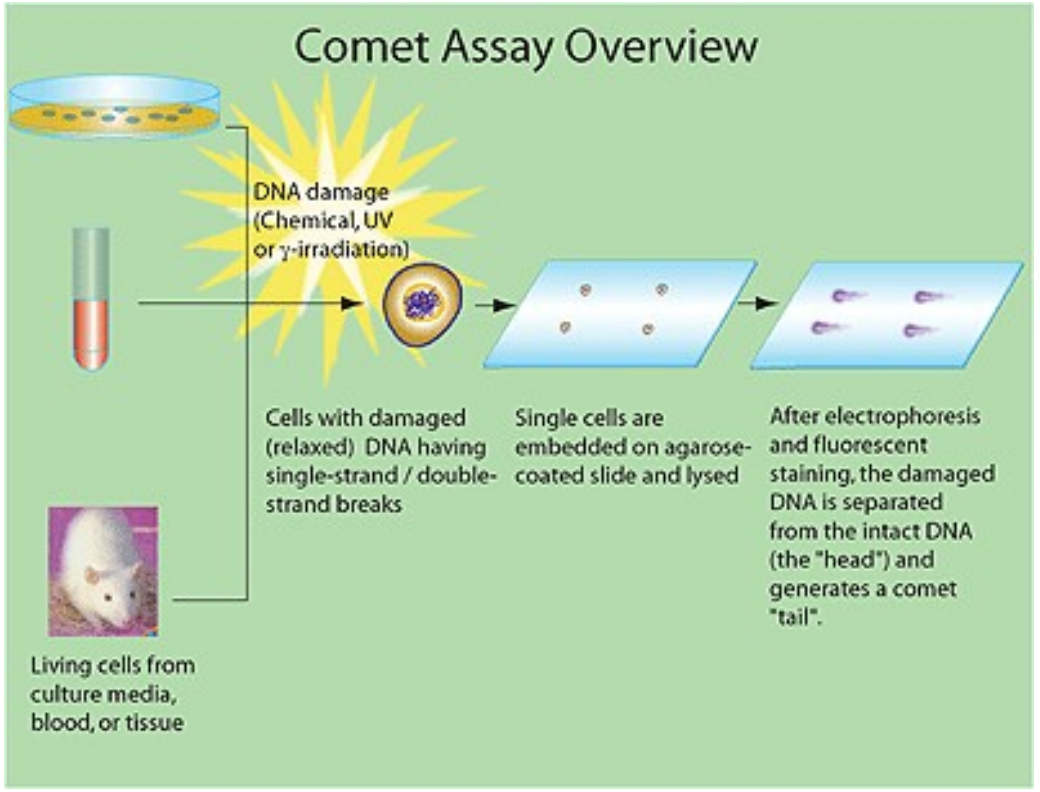
Plusieurs **dizaines ou centaines** d'échantillons à analyser :

Données : 10 000 * 2 * **10 ou 100** *« spots »*

Domaines concernés



Analyse d'image : Comet Assay



Déterminer le nombre
et la taille des « comètes »

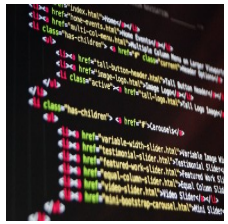
Effectuer cette analyse sur x échantillons

Faire une analyse quantitative et qualitative

→ Utilisation de ImageJ / Fidji (python / javascript)

https://www.sigmaaldrich.com/FR/fr/technical-documents/technical-article/research-and-disease-areas/cancer-research/comet-assay

Domaines concernés



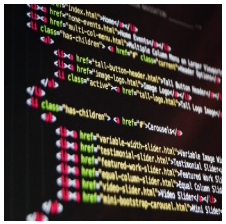
Et bien d’autres questions ...

Écologie (projet TARA) : graphes de co-occurence

Biodiversité : analyse images satellites

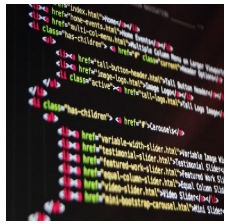
Nouvelles molécules chimiques (Roundup / glyphosate)

Défis nouveaux (**à vous de jouer**)



Programmer, difficile ?

Programmer : de la méthode !



Il est essentiel de connaître **certains algorithmes** même pour des problèmes qui peuvent paraître simple. Ces problèmes ont le plus souvent déjà été traités mais ne sont pas (encore) connus ou diffusés à grande échelle en biologie.

S'il est nécessaire d'écrire de nouvelles méthodes de traitement, il est possible de se diriger vers la **bioinformatique**, cependant la plupart des traitements en biologie font appel à des traitements mathématiques fondamentaux et à une analyse de données poussée (statistiques) **qui sont déjà implémentés**.

Pour programmer, il faut respecter plusieurs étapes :

1 - spécification du problème

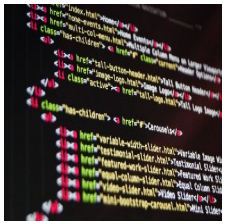
2 - analyse du problème

3 - mise en place d'un algorithme

4 - traduction dans un langage informatique

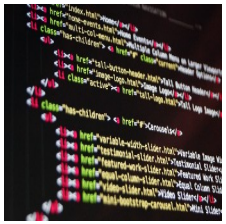
5 - tests et mise au point du programme sur machine

(1) Spécification et (2) analyse du problème



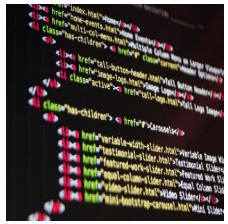
- Les spécifications servent à déterminer les objectifs à atteindre, les questions sous-jacentes, et les étapes à réaliser pour obtenir un résultat
- L'analyse permet d'écrire brièvement en français les blocs fonctionnels qui doivent être réalisés :
  - données nécessaires
  - traitement à réaliser
  - résultats attendus

(3) Mise en place d'un algorithme



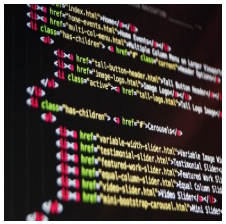
- Les étapes préliminaires permettent d'identifier les étapes faciles à réaliser pour déterminer les problèmes à résoudre
- Exemple
  - Lire un fichier de sortie d'un automate
  - Appliquer un traitement statistique, une normalisation
  - Effectuer des comparaisons par rapport à un échantillon de référence
  - Écrire le résultat sous forme de rapport

(3) Mise en place d'un algorithme



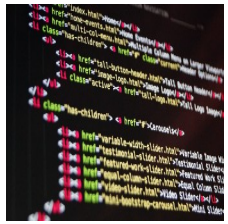
- Lire un fichier de sortie d'un automate
  - Étape **facile**, le format de sortie est probablement identifié au préalable (fichier texte, excel, etc)
- Appliquer un traitement statistique, une normalisation
  - Étape **difficile**, il faut appliquer un traitement spécifique en fonction de la taille de l'échantillon, en fonction du type de données
  - Il faut déterminer ce que l'on veut tester

(3) Mise en place d'un algorithme



- Effectuer des comparaisons par rapport à un échantillon de référence
  - Étape « **facile** », le traitement étant déjà réalisé auparavant, il suffit d'effectuer une opération mathématique simple, par exemple une soustraction
- Écrire le résultat sous forme de rapport
  - Cette étape est disponible dans de nombreux langages, il faut cependant indiquer les éléments qui sont attendus et le format du fichier (excel, word, pdf, html, …)

(4) Traduction en langage informatique

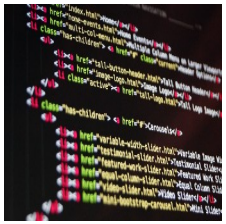


- Il va falloir écrire les instructions informatiques (**le « code »**) pour expliquer à l'ordinateur ce que l'on veut faire
- Il existe de nombreux langages avec différentes philosophies, on parle de « paradigmes de programmation »
- Nous allons nous concentrer sur un langage interprété très répandu, le Javascript.

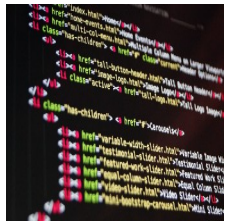
https://fr.wikipedia.org/wiki/Paradigme_(programmation)

https://www.w3schools.com/js/

(5) Test et mise au point



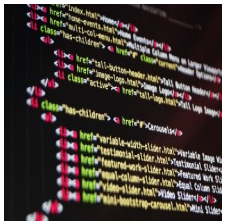
- Cette étape permet de vérifier que le code écrit permet bien d'effectuer le traitement escompté
- La mise en pratique du programme permet de se rendre compte des « cas limites » auxquels on n'avait pas pensé
- Une fois le code stabilisé on peut le mettre à la disposition des autres
- Un bon code contient des instructions sur son utilisation et de la documentation de son fonctionnement, parfois sous forme de publication (article scientifique).



Les piliers de l’algorithmique

les variables
2 types de tests
3 types de boucles
les fonctions

Conditionnelle / test "Si … Sinon Si … Sinon … Finsi"



- Algorithme

```
Si condition Alors
   Instruction 1

Sinon
   Instruction 2
Finsi
```

- Javascript

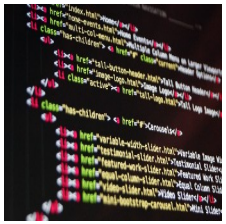
```
if (a < b ) {
     alert("a plus petit") ;
}
else {
     alert("b plus petit") ;
}
```

Formalisme du français dans l'algorithme

Vocabulaire + syntaxe normalisés pour le javascript :
- accolades pour délimiter un bloc logique { … }
- point virgule pour terminer une instruction

https://www.w3schools.com/js/js_if_else.asp

Conditionnelle / test "Selon … Cas"



- Algorithme

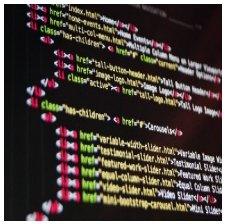
```
Selon sélecteur Faire
  Cas 1 : instruction 1


  Cas 2 : instruction 2
  Cas 3 : instruction 3
  Cas 4 : instruction 4
  Cas * : instruction 5



Finselon
```

- Javascript

```
switch valeur {
    case 1:
        alert("cas 1");
        break;
    case 2:
        alert("cas2");
        break;
    Case *:
        alert("Tous les
autres cas");
        break;
}
```

Structure utile quand il n'y a qu'**un seul élément** dont la valeur change, évite d'avoir à enchaîner beaucoup de if, else if, ...

La boucle "Pour"



- Algorithme

```
Pour condition Faire
  Instruction
FinPour
```

- Javascript

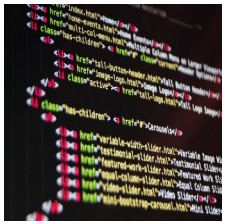
```
for (i=0 ; i < 10 ; i+1 ) {
    alert("a vaut " + i);
}
```

La boucle est exécutée **sur un nombre d'étapes défini a priori :**

i=0 : le compteur « i » est initialisé à zéro au départ
i < 10 : tant que i est inférieur à 10, la boucle est exécutée
I+1 : on incrémente à chaque tour de boucle le compte i de 1

Cette boucle **est utile quand le nombre d'itérations à réaliser est connu, ou peut être déterminé à l'avance**

La boucle "Tant Que"



- Algorithme

```
TantQue condition Alors
  Instruction
FinTantQue
```

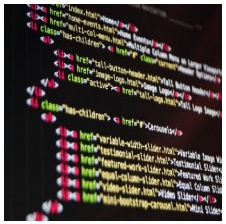
- Javascript

```
while (a < b ) {
    alert("a plus petit") ;
}
```

La boucle est exécutée **tant que la condition est remplie**,
il faut donc avoir une instruction qui met à jour le résultat de la condition,
sinon la boucle est infinie.

La boucle **peut ne peut s'exécuter** si la condition n'est pas remplie initialement

La boucle "Jusqu'à"



- Algorithme

```
Faire
  Instruction
Jusqu'à condition
```

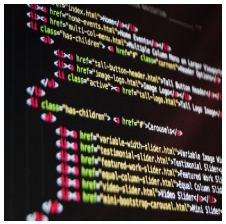
- Javascript

```
do {
    alert("a plus petit") ;
} while (a < b );
```

La boucle est exécutée **tant que la condition est remplie**,
il faut donc avoir une instruction qui met à jour le résultat de la condition,
sinon la boucle est infinie.

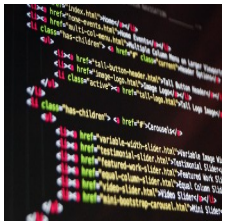
La boucle est réalisée au moins une fois.

Les fonctions



- L’objectif d’une fonction est de regrouper des éléments algorithmiques répétitifs d’un programme dans un code bien délimité
- Une fonction
  - reçoit une ou plusieurs données
  - effectue des instructions spécifiques
  - renvoie un résultat lié au traitement

Les fonctions



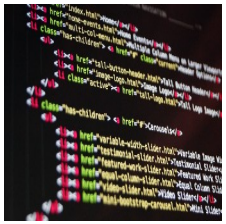
- Algorithme

```
fonction nom (paramètres) :
  Valeur interne <- param1
  Instruction
  retourne resultat
Finfonction
// Appel de la fonction
D = nom(a, b)
```

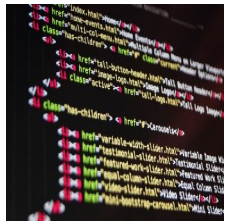
- Javascript

```
function multiplie (x, y) {
    let interne;
    interne = x * y ;
    return interne;
}
// Appel de la fonction
var r = multiplie(4,5);
```

Les fonctions

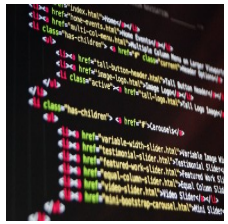


- Une variable déclarée dans une fonction n'existe que le temps de son utilisation, elle est détruite après l'exécution de la fonction.
  On parle de **variable LOCALE (let)**.
- Si la variable est déclarée en dehors d'une fonction, alors il s'agit d'une **variable GLOBALE** qui existe tant que le programme est utilisé (**var**)
- Réutilisation
- Des fonctions pré-existantes sont disponibles



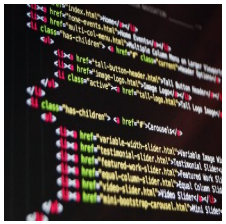
Représentation des données

L’unité d’information élémentaire, le bit

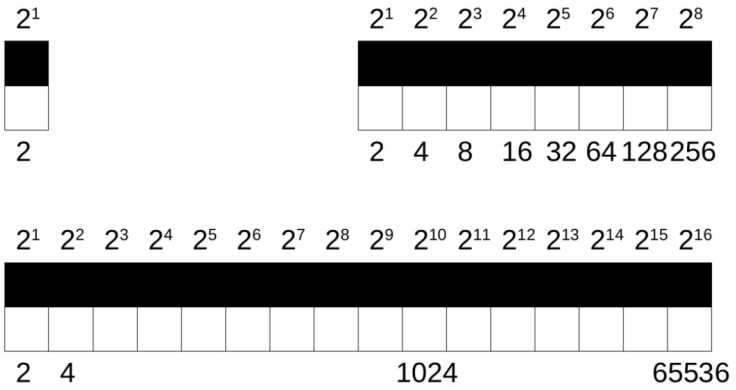


- La représentation d’un signal a deux états
  - Allumé (1)
  - Éteint (0)
- On parle d’encodage **binaire**
- Problème : seulement deux informations peuvent être représentées ainsi
  - Allumé ou éteint
  - Vacciné ou non vacciné
  - Yin / Yang

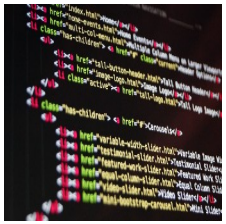
Représentation des informations



- Pour stocker plus d'information, il faut regrouper les bits en « paquets » consécutifs
  - 8 bits : 2^8 = 256 possibilités, **1 octet** (chromosome)
  - 16 bits : 2^{16} = 65 536 possibilités (gène)
  - 32 bits : 2^{32} = 4 294 967 296 (génome)
  - 64 bits : 2^{64} = 1,8447.10^{19} (protéome)

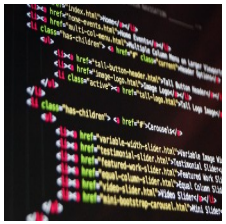


Représentation des informations



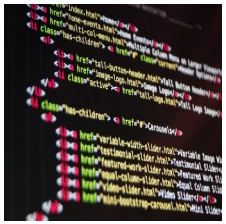
- Taille variable des informations : comment le spécifier ? Il faut « typer » les variables
  - char : initialement 8 bits (1 octet)
  - string : 16 bits (caractère « universel » UTF-8)
  - int : 32 bits (entier)
  - float : 32 bits (chiffres à virgule)
  - number : 64 bits (entiers et chiffres à virgule)
  - booléen (true / false)

Distinguer caractères et chiffres



- C’est la **notation** qui permet de distinguer les deux grands types de variables
  - Si la variable est indiquée tel quel, par défaut elle est considérée comme un **nombre** :
    // Exemple
    var lavariable = **5**;
  - Si la variable est **entourée de guillements** (anglais), alors on indique qu’il s’agit d’une **chaîne de caractères** :
    var lavariable = **"exemple"**;

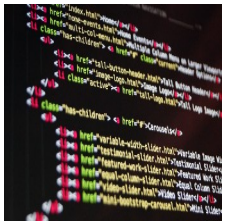
Typage et transtypage



- Dans certains langages, il faut indiquer explicitement le type de la donnée (int, float, number, etc). **Python et javascript font cette opération de manière transparente**.
- Attention aux erreurs cependant avec l'opération « + » (addition ou **concaténation**)

```
var a = 5;
var b = "dutexte";
var c = 4;
var h = a + c ; (h vaut 4+5 soit 9)
var g = b + c ; (g vaut "dutexte4")
var k = a + b ; (normalement erreur, sinon "5dutexte")
```

Regrouper plusieurs variables

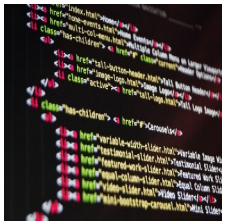


- Pour regrouper les différents variables, on va utiliser des listes ou des tableaux associatifs
  - **Listes** : suite d'éléments d'un même type, ex :
    var l = [20,2,33,45,6,7,1000000]; (nombres)
    var m = ["annee", "prenom", "nom", "age"]; (texte)
  - **Tableaux associatifs** (système clé / valeur) :

```
var d = {"annee":2000,
          "age":22,
          "promotion" : "L2"
        } ;
```

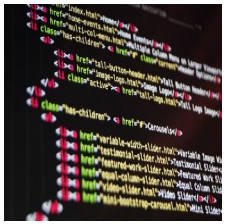
- **Attention, implémentation et noms différents selon les langages**

Données complexes



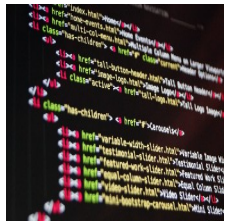
- Quand il est nécessaire de regrouper des données de différents types, et qu'il faut en plus appliquer un traitement pour utiliser ces données, on a recours à un **objet**.
- En javascript, on l'écrira ainsi :
  var **d** = **new Date()**
- L'objet contient une seule information, le nombre de millisecondes écoulées depuis le 1er janvier 1970, et les fonctions qui permettent d'afficher la date dans différentes langues, fuseaux horaires, horaire d'été ou d'hiver, etc.

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Date



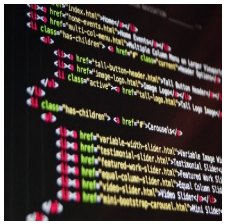
Mettre en pratique la programmation à travers un navigateur web

Un « site web »



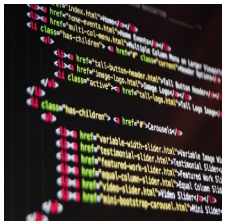
- Une page internet est un fichier au **format texte, avec un éditeur de TEXTE** (*pas Word ou LibreOffice*) qui comprend le contenu de la page et des instructions d'affichage interprétées par un navigateur
- Ces instructions sont écrites en HTML, la mise en page est décrite avec du CSS, et les traitements liés aux actions de l'utilisateur sont écrits en JS - JavaScript (*parti pris*)
- L'accès au fichier se fait via internet http://monsite.org, ou en ouvrant ouvrir un fichier local : \\Utilisateur\nom\monsite.html
- Chaque langage (HTML, CSS, JS) a une syntaxe propre !

Hyper Text Markup Langage

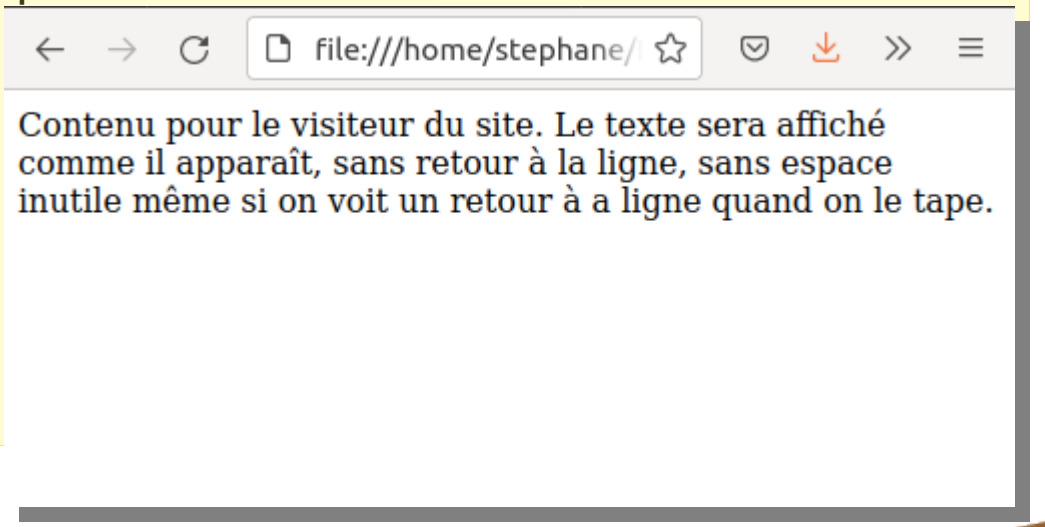


- Le HTML appartient à la méta famille des langages « à balises » (*markup* en anglais).
- Les blocs logiques sont définis par des mot-clés entourés par < et >.
  - <html> … </html>
  - <head> … </head>
- La plupart du temps les balises sont ouvrantes (< … >) et fermantes (</ … >) : il faut un couple de balises pour déterminer un bloc logique
- Cette syntaxe par balise est très utilisée en biologie et dans les traitements de texte (fichiers **XML**).

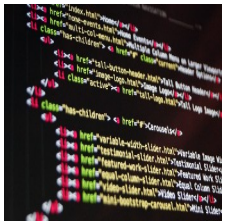
Structure minimale d'un fichier HTML



```
<!DOCTYPE html>
<html>
    <head>
        <!-- Contenu destiné au navigateur -->
        <meta charset="utf-8"/>
        <script>
        // Programme informatique en JS
        </script>
    </head>
    <body>
      Contenu pour le visiteur du site. Le texte sera affiché
      comme il apparaît, sans
      retour à la ligne, sans espace inutile

      même si on voit

      un retour à a ligne


      quand on le tape.
    </body>
</html>
```



Organisation d’une page web

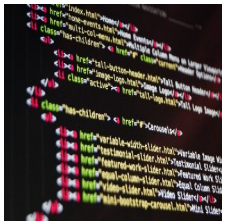


- L’affichage de tous les éléments d’une page se fait par défaut les uns à la suite des autres :
  - Mise en page « en ligne » : texte simple, image
- Certaines balises forcent le retour à la ligne après leur fermeture, on parle de
  - Mise en page « en bloc » : tableaux, listes

https://developer.mozilla.org/fr/docs/Web/HTML/Inline_elements

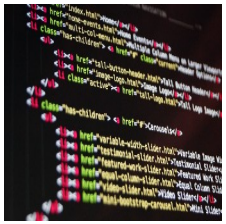
https://developer.mozilla.org/fr/docs/Web/HTML/Block-level_elements

Structuration du document : titres et texte



- Chaque élément de texte doit être inclus dans des balises « paragraphe » : …
- Un paragraphe est de type « **en bloc** »
  ** **Ceci est un paragraphe.** **
- Les titres sont définis avec la balise « *header* », et un chiffre qui désigne le niveau hiérarchique :
  - **<h1>**Titre de niveau 1**</h1>**
  - **<h2>**Titre de niveau 2**</h2>**
  - **<h3>**Titre de niveau 3**</h3>**
  - **…**
  - **<h6>**Titre de niveau 6, dernier prévu par défaut**</h6>**
- Retour à la ligne forcé avec la balise auto-fermante **
**

Tableaux (!)



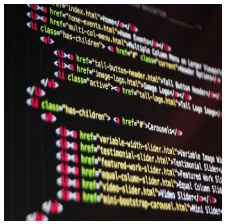
```
<table>
    <tr>
        <td>Cellule 1</td><td>Cellule 2</td>
    </tr>
    <tr><td>Cellule 1</td><td>Cellule 2</td></tr>
</table>
```

tr : Table Row (ligne du tableau)

td : Table Data (cellule du tableau)

(!) Version simplifiée d’un tableau en HTML5

Listes



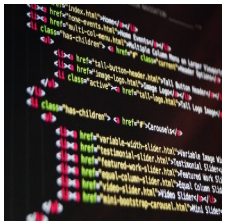
```
<ul> (ou <ol>)
    <li>
        Ceci est le premier élément de la liste
    </li>
    <li>Deuxième élément de la liste</li>
</ul> (ou </ol>)
```

li : List Item (élément de la liste)

ul : Unordered List (liste à puces)

ol : Ordered List (liste numérotée)

Images



```
<img src="image.png" alt="texte qui est affiché si l'image n'est pas trouvée" id="image1" width=300 height=200>
```

1) src : attribut pour indiquer le fichier de l'image à afficher

2) alt : un texte qui s'affiche si l'image n'est pas trouvée

3) id : attribut qui permet d'identifier de manière unique une image

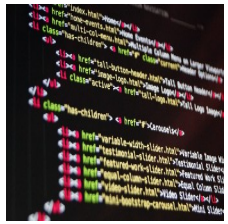
4) width et/ou height (au moins un des deux) : dimension de l'image à afficher

Les attributs de l'image DOIVENT être précisés (4 attributs).

La taille de l'image peut être spécifiée en pixels (par défaut), en cm, en pourcentage, etc. Dans ce cas il faut préciser l'unité.

Si une seule des dimensions de l'image est précisée, alors le navigateur fera un redimensionnement proportionnel.

Liens

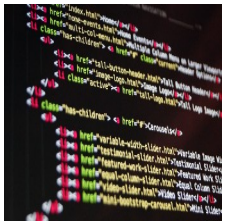


- La balise « **a** » (*anchor*) sert à définir un lien. Le cas général est un lien qui renvoie vers une autre page, par exemple sur le site de l'université. On parle alors de **lien externe**.
  - **<a** href="https://www.univ-nantes.fr"**>**Lien vers l'université de Nantes****
- Il est possible de faire référence à un élément dans la page, il faut alors 2 balises ancres
  - 1) une balise à l'endroit que l'on veut référencer

    ```
    <a id="endroit"></a>
    ```
  - 2) une balise ailleurs qui permet de se déplacer rapidement dans la page

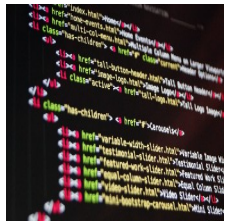
    ```
    <a href="#endroit">Aller à l'endroit</a>
    ```

Organisation générale d'une balise



- Une balise peut comprendre **un ou plusieurs attributs** avec une valeur associée. Certains sont obligatoires, beaucoup sont facultatifs.
  - **<a** id="lelien"**>**Cliquez ici****
  - **<h1** id="haut"**>**
  - **<td** id="cellule1a" width=100%**>**Cellule 1a**</td>**
- Les balises **doivent** être imbriquées dans l'ordre inverse de leur ouverture.
  - ** **Le texte contient un lien vers une page externe ** **.

En TD et en TP



- En TD : exercices liés à des problèmes algorithmiques « simples » **utiles en biologie**
- En TP : mise en place de la page au format HTML, ajout des éléments javascript nécessaires, utilisation (a minima) de CSS
- **HTML** + CSS : séparer le **fond** de la *forme*

http://www.csszengarden.com/