M1 MIAGE Option RIM
Cours 1 : Introduction à la recherche d'informations dans des données

Alexandre Termier

2010-2011 S2

Présentation de l'option RIM

- Organisation
  - Lundi : 1h30 CM
  - Mardi : 1h30 TP/TD
- Evaluation
  - CC : 1/3
  - Examen terminal : 2/3
- Règles
  - Présence
  - Participation
- Contact

  Alexandre.Termier@imag.fr

Les données à travers les âges

- Préhistoire
  - Compétences vitales
  - Lieux proches
  - Transmission orale
  - Echanges limités
- Moyen-Age (pré-Guttenberg)
  - Beaucoup de livres
  - Accès : nobles et religieux
  - Diffusion européenne
- Peu de données

- Post-Guttenberg $\rightarrow$ 2000
  - Livres, journaux...
  - Accessibles au plus grand nombre
  - Diffusion mondiale
  - Peu de recherche
- Ordinateur + Internet
  - Collecte automatique de grand volumes de données
  - Stockage gigantesque
  - Transmissions rapides
  - Recherche efficace
  - Trop d'informations, pas assez de savoir

Des données récentes

- Pages web > 1000 milliards (Google 2008)
- Génome humain
  - 3.4 milliards paires de bases (3 GB)
  - 25000 gènes
  - Annotations
- AT & T
  - 312 TB dans **une** base de données (2005)
  - 2800 milliards d'appels téléphoniques
- ChoicePoint
  - 250 TB sur 220 millions d'Américains
- Word Data Center for Climate (WDCC)
  - 220 TB accès direct + 6 TB bandes magnétiques
  - Données climatiques réelles + simulations de paléoclimats

Problèmes avec ces données

Problème 1

- Comment stocker les données ?
- Comment y accéder efficacement ?

→ Bases de données "traditionnelles"

Problème 2

- Comment convertir ces grands volumes de données en **connaissances utiles** ?

→ Data Mining

Qu'est-ce que le Data Mining ?

- Une partie du processus de **KDD : Knowledge Discovery in Data**

Definition (Fayyad, 1996)

KDD is to apply computational techniques to *identify valid, novel, potentially usefull and ultimately understandable patterns in data*.

- Processus de KDD
  1. Données brutes
  2. Pré-processing
  3. Data Mining
  4. Post-processing
  5. Information

Plan du cours

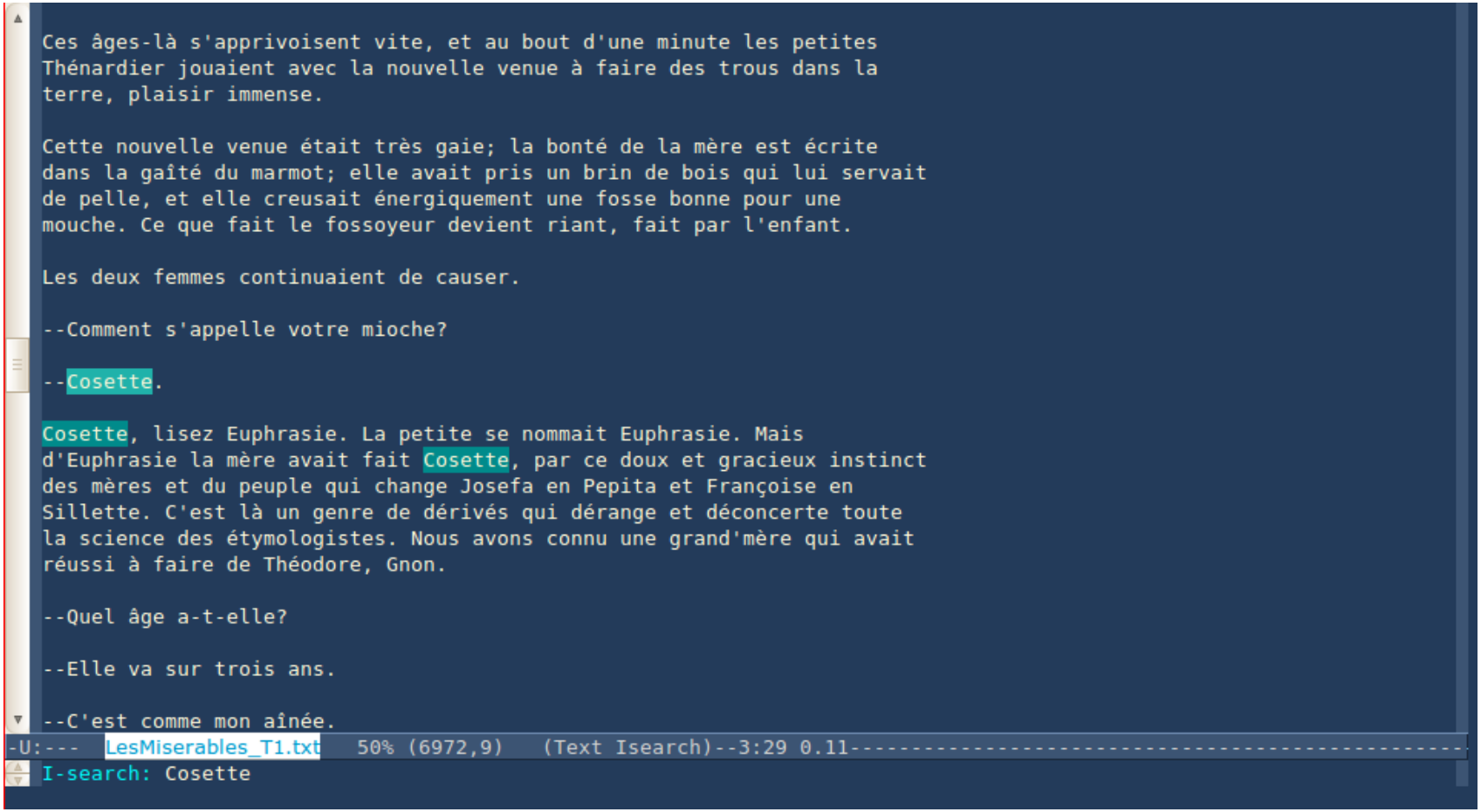
- Matching simple
- Preprocessing des données
- Patterns fréquents
- Classification
- Clustering

Invité du cours

Pattern matching

- Où commence l'analyse de données ?
- $\rightarrow$ quand on cherche quelque chose dans les données
- **find**

Find GUI



- Problèmes
  - Vue d'ensemble de toutes les occurrences ?
  - Traitements sur les résultats ?

Find en ligne de commande

- Commande unix `grep` : `grep texte fichier(s)`
  - `grep` lit les lignes du/des fichier(s) une par une
  - il affiche les lignes contenant `texte` (= *match*)

Example (grep simple)

```
$ grep Valjean *.txt
LesMiserables_T1.txt:Chapitre VI Jean Valjean
LesMiserables_T1.txt:nom? Vous vous appelez Jean Valjean. Maintenant voulez-vous que je vous
LesMiserables_T1.txt:--Voici. Je m'appelle Jean Valjean. Je suis un galérien. J'ai passé
LesMiserables_T1.txt:qu'on a mis sur le passeport: «Jean Valjean, forçat libéré, natif
LesMiserables_T1.txt:«--Monsieur Jean Valjean, c'est à Pontarlier que vous allez?
LesMiserables_T1.txt:Pontarlier, où vous allez, monsieur Valjean, une industrie toute
LesMiserables_T1.txt:homme, qui s'appelle Jean Valjean, n'avait que trop sa misère présente à
LesMiserables_T1.txt:que tous les soirs, et il a soupé avec ce Jean Valjean du même air et de
LesMiserables_T1.txt:Jean Valjean
LesMiserables_T1.txt:Vers le milieu de la nuit, Jean Valjean se réveilla.
...
```

Quelques options de grep (1/3)

- `-i` : ignore la casse

Example (grep -i)

```
$ grep -i merde *.txt
LesMiserables_T2.txt:Français, rendez-vous!_ Cambronne répondit: _Merde!_
```

- `-c` : compte des lignes des fichiers qui matchent

Example (grep -c)

```
$ grep -c Valjean *.txt
LesMiserables_T1.txt:191
LesMiserables_T2.txt:273
LesMiserables_T3.txt:1
LesMiserables_T4.txt:2
LesMiserables_T5.txt:444
```

Quelques options de grep (2/3)

- `-e texte` : autre façon de spécifier ce qu'on cherche, répétable

Example (grep -e)

```
$ grep -e Valjean -e Cosette *.txt
...
LesMiserables_T1.txt:--Jean Valjean! il y aura autour de toi beaucoup de voix qui feront un
LesMiserables_T1.txt:Jean Valjean, qui a été maire à Montreuil-sur-mer_! Le soir, ruisselant
LesMiserables_T1.txt:autrefois elle endormait sa petite Cosette, et qui ne s'était pas
LesMiserables_T1.txt:--Parti! s'écria-t-elle. Il est allé chercher Cosette!
...
```

- `-n` : rajoute les numéros de ligne

Example (grep -n)

```
$ grep -n -e Valjean -e Cosette *.txt
...
LesMiserables_T1.txt:10805:--Jean Valjean! il y aura autour de toi beaucoup de voix qui feront un
LesMiserables_T1.txt:10863:Jean Valjean, qui a été maire à Montreuil-sur-mer_! Le soir, ruisselant
LesMiserables_T1.txt:11803:autrefois elle endormait sa petite Cosette, et qui ne s'était pas
LesMiserables_T1.txt:11871:--Parti! s'écria-t-elle. Il est allé chercher Cosette!
...
```

Quelques options de grep (3/3)

- `-l` : noms des fichiers contenant des matchs
- `-L` : noms des fichiers ne contenant pas de matchs

Example (grep -l / grep -L)

```
$ grep -l Myriel *.txt
LesMiserables_T1.txt

$ grep -L Myriel *.txt
LesMiserables_T2.txt
LesMiserables_T3.txt
LesMiserables_T4.txt
LesMiserables_T5.txt
```

Vers un matching plus fin

- `grep '^texte' fichier(s)` : cherche texte uniquement en début de ligne

Example (1)

```
$ grep '^Valjean' *.txt
LesMiserables_T1.txt:Valjean qu'une soeur plus âgée que lui, veuve, avec sept enfants, filles
LesMiserables_T1.txt:Valjean. Sa jeunesse se dépensait ainsi dans un travail rude et mal
...
```

- `grep 'texte$' fichier(s)` : cherche texte uniquement en fin de ligne

Example (2)

```
$ grep 'Valjean$' *.txt
LesMiserables_T1.txt:Chapitre VI Jean Valjean
LesMiserables_T1.txt:Jean Valjean
```

Vers un matching plus fin

- `grep '^texte' fichier(s)` : cherche texte uniquement en début de ligne

Example (1)

```
$ grep '^Valjean' *.txt
LesMiserables_T1.txt:Valjean qu'une soeur plus âgée que lui, veuve, avec sept enfants, filles
LesMiserables_T1.txt:Valjean. Sa jeunesse se dépensait ainsi dans un travail rude et mal
...
```

- `grep 'texte$' fichier(s)` : cherche texte uniquement en fin de ligne

Example (2)

```
$ grep 'Valjean$' *.txt
LesMiserables_T1.txt:Chapitre VI Jean Valjean
LesMiserables_T1.txt:Jean Valjean
```

Example (1 et 2)

```
$ grep '^Valjean$' *.txt
termier@Osaka:~/Enseignement/2010_2011/RIM/Data$ grep '^Jean Valjean$' *.txt
LesMiserables_T1.txt:Jean Valjean
```

Comment matcher plusieurs mots différents ?

- Parfois on veut plus qu'un mot précis, on veut un ensemble de mots qui suivent une "règle" commune

Example (enfants)

Comment matcher à la fois :

- enfant
- enfants
- Enfant
- Enfants

avec grep ?

Comment matcher plusieurs mots différents ?

- Parfois on veut plus qu'un mot précis, on veut un ensemble de mots qui suivent une "règle" commune

Example (enfants)

Comment matcher à la fois :

- enfant
- enfants
- Enfant
- Enfants

avec grep ?

- Ici on a : `[E ou e] nfant [s ou rien]`

Autre exemple

Example (Numéros de téléphone)

- 01-11-22-33-44
- 04.76.82.72.07
- 02 77 88 99 00
- 095566432288

Autre exemple

Example (Numéros de téléphone)

- 01-11-22-33-44
- 04.76.82.72.07
- 02 77 88 99 00
- 095566432288

- Forme :
  - `0[chiffre entre 1 et 9]`
  - `[tiret ou espace ou point ou rien]`
  - `[chiffre entre 0 et 9]` deux fois
  - `[tiret ou espace ou point ou rien]`
  - `[chiffre entre 0 et 9]` deux fois
  - `[tiret ou espace ou point ou rien]`
  - `[chiffre entre 0 et 9]` deux fois
  - `[tiret ou espace ou point ou rien]`
  - `[chiffre entre 0 et 9]` deux fois

Les expressions rationnelles (regular expressions)

Definition (Expression rationelle)

Une expression régulière est une chaîne de caractères particulière décrivant un **motif**. Ce motif représente un ensemble de chaînes de caractères.

[ ]	Matche un des caractères entre [ ]
-	Décrit un domaine de valeurs : [0-9] = les chiffres
[^]	Matche n'importe quel caractère sauf ceux dans [^]
\	Annule la signification spéciale du caractère suivant \^ = ^
.	Matche n'importe quel caractère sauf fin de ligne
?	L'expression précédente est matchée 0 ou 1 fois
+	L'expression précédente est matchée 1 ou plusieurs fois
*	L'expression précédente est matchée 0 ou plusieurs fois
$\{n\}$	L'expression précédente est matchée exactement n fois
$\{n, m\}$	L'expression précédente est matchée entre n et m fois
()	Groupe une expression
$expr_1 \mid expr_2$	Matche $expr_1$ ou $expr_2$

Exemples d'expressions rationnelles

Example (enfants)

L'expression est : `[Ee]nfants?`

```
$ egrep '[Ee]nfants?' *.txt
LesMiserables_T1.txt:_Pour les enfants trouvés: cinq cents livres_
LesMiserables_T1.txt:une femme et pour l'enfant qu'il avait d'elle, à bout de ressources,
LesMiserables_T1.txt:Enfants-Rouges, où l'eau arrive par des robinets. On n'est pas dans le
LesMiserables_T2.txt:formule varie. Ainsi, à l'Enfant-Jésus, on dit:--_À l'heure qu'il est et
```

Exemples d'expressions rationnelles

Example (enfants)

L'expression est : `[Ee]nfants?`

```
$ egrep '[Ee]nfants?' *.txt
LesMiserables_T1.txt:_Pour les enfants trouvés: cinq cents livres_
LesMiserables_T1.txt:une femme et pour l'enfant qu'il avait d'elle, à bout de ressources,
LesMiserables_T1.txt:Enfants-Rouges, où l'eau arrive par des robinets. On n'est pas dans le
LesMiserables_T2.txt:formule varie. Ainsi, à l'Enfant-Jésus, on dit:--_À l'heure qu'il est et
```

Example (téléphones)

L'expression est : `0[1-9]([ .]?[0-9]{2}){4}`

```
$egrep '0[1-9]([ .\-]?[0-9]{2}){4}' CM1_exTelSmall.txt
01-11-22-33-44
04.76.82.72.07
02 77 88 99 00
095566432288
```

Exemples d'expressions rationnelles

Example (enfants)

L'expression est : `[Ee]nfants?`

```
$ egrep '[Ee]nfants?' *.txt
LesMiserables_T1.txt:_Pour les enfants trouvés: cinq cents livres_
LesMiserables_T1.txt:une femme et pour l'enfant qu'il avait d'elle, à bout de ressources,
LesMiserables_T1.txt:Enfants-Rouges, où l'eau arrive par des robinets. On n'est pas dans le
LesMiserables_T2.txt:formule varie. Ainsi, à l'Enfant-Jésus, on dit:--_À l'heure qu'il est et
```

Example (téléphones)

L'expression est : `0[1-9]([ .]?[0-9]{2}){4}`

```
$egrep '0[1-9]([ .\-]?[0-9]{2}){4}' CM1_exTelSmall.txt
01-11-22-33-44
04.76.82.72.07
02 77 88 99 00
095566432288
```

Example (Valjean ET Cosette)

```
$ egrep '(Valjean.*Cosette)|(Cosette.*Valjean)' *.txt
LesMiserables_T2.txt:Le soir même du jour où Jean Valjean avait tiré Cosette des griffes des
LesMiserables_T2.txt:de Cosette cherchait un père comme l'instinct de Jean Valjean cherchait
...
```

Limites de grep

- Formattage limité des résultats
- Pas d'opérations possibles à part le comptage

Example (Fournisseurs du Bedford Borough Council, data.gov.uk)

```
$ grep Mouchel BedfordBoroughCouncil_July2010.csv
"Childrens Services, Schools & Families":"Mouchel":10132563:8527.69
"Childrens Services, Schools & Families":"Mouchel":10130857:9645.46
"Childrens Services, Schools & Families":"Mouchel":10136214:9051.05
"Childrens Services, Schools & Families":"Mouchel":10130291:4078.31
"Childrens Services, Schools & Families":"Mouchel":10130291:1137.91
"Childrens Services, Schools & Families":"Mouchel":10130857:1729.32
"Childrens Services, Schools & Families":"Mouchel":10130282:9703.49
"Finance & Corporate Services":"Mouchel":10130866:1465.86
"Finance & Corporate Services":"Mouchel":10130866:3331.50
"Finance & Corporate Services":"Mouchel":10130866:2473.31
"Finance & Corporate Services":"Mouchel":10130866:1973.02
"Finance & Corporate Services":"Mouchel":10130866:1465.86
"Finance & Corporate Services":"Mouchel":10130869:1975.95
"Finance & Corporate Services":"Mouchel":10136206:1037.27
```

Traitement de fichiers textes

- Plusieurs langages de "script" pour traiter les fichiers textes:
  - `awk`
  - `sed`
  - `Perl`
  - `(Python)`
  - `(Ruby)`
  - `...`
- Souvent plus simples à utiliser que les langages traditionnels (C,Java,...)
- Scripts avec input=texte, output=texte sont (souvent) composables

AWK

Crée au Bell Labs en 1977 - **A** ho, **W** einberger, **K** ernnighan.
Standard sur n'importe que système UNIX.

- Structure d'un programme awk

```
BEGIN { ...instructions... }

/regexp1/ { ...instructions... }

...

/regexpN/ { ...instructions... }

END   { ...instructions... }
```

Fonctionnement de awk

- Effectue les instructions de `BEGIN`
- Lit le fichier d'entrée ligne par ligne
  - Pour chaque ligne cherche toutes les regexp qui matchent
  - Effectue les instructions des regexp matchées
- Effectue les instructions de `END`

Programmation en awk

- Contenu des `{ ...instructions... }`
  - initialisations/modifications de variables
  - `if ... then ... else ...`
  - `while...` et `for...`
- Variables spéciales
  - accès aux champ *i*: `$i`
  - `NF`: **N** umber of **F** ields
  - `NR`: **N** umber of **R** ows
- Appel
  - `awk programme.awk fichier`
  - `awk 'code awk' fichier`
  - option `-F` pour déterminer le séparateur de champs

Exemple awk : formattage

Champs des données

“Childrens Services, Schools & Families”	“Mouchel”	10132563	8527.69
$1	$2	$3	$4

Example (Formattage avec awk)

```
$ awk -F: '/Mouchel/ {print "Mouchel supplied £",$4,"in category",$1}'
          BedfordBoroughCouncil_July2010.csv
```

Exemple awk : formattage

Champs des données

“Childrens Services, Schools & Families”	“Mouchel”	10132563	8527.69
$1	$2	$3	$4

Example (Formattage avec awk)

```
$ awk -F: '/Mouchel/ {print "Mouchel supplied £",$4,"in category",$1}'
          BedfordBoroughCouncil_July2010.csv


Mouchel supplied £ 8527.69 in category "Childrens Services, Schools & Families"
Mouchel supplied £ 9645.46 in category "Childrens Services, Schools & Families"
Mouchel supplied £ 9051.05 in category "Childrens Services, Schools & Families"
Mouchel supplied £ 4078.31 in category "Childrens Services, Schools & Families"
Mouchel supplied £ 1137.91 in category "Childrens Services, Schools & Families"
Mouchel supplied £ 1729.32 in category "Childrens Services, Schools & Families"
Mouchel supplied £ 9703.49 in category "Childrens Services, Schools & Families"
Mouchel supplied £ 1465.86 in category "Finance & Corporate Services"
Mouchel supplied £ 3331.50 in category "Finance & Corporate Services"
Mouchel supplied £ 2473.31 in category "Finance & Corporate Services"
Mouchel supplied £ 1973.02 in category "Finance & Corporate Services"
Mouchel supplied £ 1465.86 in category "Finance & Corporate Services"
Mouchel supplied £ 1975.95 in category "Finance & Corporate Services"
Mouchel supplied £ 1037.27 in category "Finance & Corporate Services"
```

Exemple awk : calcul de somme

Example (Calcul avec awk)

```
awk -F: '/Mouchel/ {print "£",$4,"bought from Mouchel in category",$1 ; sum += $4}
         END { print "\nTotal bought from Mouchel: £",sum}'
         BedfordBoroughCouncil_July2010.csv
```

Exemple awk : calcul de somme

Example (Calcul avec awk)

```
awk -F: '/Mouchel/ {print "£",$4,"bought from Mouchel in category",$1 ; sum += $4}
         END { print "\nTotal bought from Mouchel: £",sum}'
         BedfordBoroughCouncil_July2010.csv


£ 8527.69 bought from Mouchel in category "Childrens Services, Schools & Families"
£ 9645.46 bought from Mouchel in category "Childrens Services, Schools & Families"
£ 9051.05 bought from Mouchel in category "Childrens Services, Schools & Families"
£ 4078.31 bought from Mouchel in category "Childrens Services, Schools & Families"
£ 1137.91 bought from Mouchel in category "Childrens Services, Schools & Families"
£ 1729.32 bought from Mouchel in category "Childrens Services, Schools & Families"
£ 9703.49 bought from Mouchel in category "Childrens Services, Schools & Families"
£ 1465.86 bought from Mouchel in category "Finance & Corporate Services"
£ 3331.50 bought from Mouchel in category "Finance & Corporate Services"
£ 2473.31 bought from Mouchel in category "Finance & Corporate Services"
£ 1973.02 bought from Mouchel in category "Finance & Corporate Services"
£ 1465.86 bought from Mouchel in category "Finance & Corporate Services"
£ 1975.95 bought from Mouchel in category "Finance & Corporate Services"
£ 1037.27 bought from Mouchel in category "Finance & Corporate Services"

Total bought from Mouchel: £ 57596
```

Limites de awk

- Pas conçu pour faire de gros programmes
  - Par ex. pas de fonctions $\rightarrow$ amélioré dans `nawk`
- Structure de traitement figée
  - Base = traitement ligne par ligne
  - Que faire pour des propriétés qui peuvent s'étendre sur plusieurs lignes ?
  - Ex.: *Listes des mots différents suivant le mot X ?*

Traitements complexes

- On doit utiliser un langage de programmation généraliste
- Dans ce cours : Haskell
- Haskell : Langage fonctionnel **pur** et **paresseux**
- Pourquoi ?
  - Très expressif : programmes courts
  - Nombreuses fonctions pré-définies
  - Typage (très) fort + pur : fiable
  - Tests faciles dans l'interpréteur
  - Options d'optimisations via le compilateur
  - Parallélisme

Premier contact avec Haskell

- Notre nouvel ami : l'interpréteur Haskell
  - Execution de commandes / programmes
  - Apprendre le langage
  - Tester
  - Débugger
  - Lancé avec : `ghci`

Example (ghci)

```
$ ghci-6.12.3
GHCi, version 6.12.3: http://www.haskell.org/ghc/  :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.
Loading package ffi-1.0 ... linking ... done.
Prelude>
```

- Prelude : "Package" de base Haskell
  - Calculs simple
  - Opérations sur les listes
  - I/O de base

ghci comme calculatrice

Example (Calculs)

```
Prelude> 30000 / 125
240.0
Prelude> it * 1.186
284.64
Prelude> let yenEuro = 125.0
Prelude> 9990 / yenEuro
79.92
Prelude> let convertYenEuro prix = prix / 125
Prelude> convertYenEuro 9990
79.92
Prelude> let applyTax prix = prix * 1.186
Prelude> applyTax it
94.78511999999999
Prelude> applyTax (convertYenEuro 9990)
94.78511999999999
Prelude> applyTax $ convertYenEuro 9990
94.78511999999999
Prelude> (applyTax . convertYenEuro) 9990
94.78511999999999
```

- `let fonction paramètres = expression`
- $: application de fonction
- . : composition de fonctions (en maths : $f \circ g$)

Typage

- Haskell : langage fortement typé
- *Quand on connaît le type d'une fonction, on peut avoir une idée de ce qu'elle fait !*

Example (types de l'ex. précédent)

```
Prelude> :t yenEuro
yenEuro :: Double
```

Typage

- Haskell : langage fortement typé
- *Quand on connaît le type d'une fonction, on peut avoir une idée de ce qu'elle fait !*

Example (types de l'ex. précédent)

```
Prelude> :t yenEuro
yenEuro :: Double


Prelude> :t convertYenEuro
convertYenEuro :: (Fractional a) => a -> a
Prelude> :t applyTax
applyTax :: (Fractional a) => a -> a
```

Typage

- Haskell : langage fortement typé
- *Quand on connaît le type d'une fonction, on peut avoir une idée de ce qu'elle fait !*

Example (types de l'ex. précédent)

```
Prelude> :t yenEuro
yenEuro :: Double


Prelude> :t convertYenEuro
convertYenEuro :: (Fractional a) => a -> a
Prelude> :t applyTax
applyTax :: (Fractional a) => a -> a


Prelude> applyTax 15
17.79
```

Typage

- Haskell : langage fortement typé
- *Quand on connaît le type d'une fonction, on peut avoir une idée de ce qu'elle fait !*

Example (types de l'ex. précédent)

```
Prelude> :t yenEuro
yenEuro :: Double


Prelude> :t convertYenEuro
convertYenEuro :: (Fractional a) => a -> a
Prelude> :t applyTax
applyTax :: (Fractional a) => a -> a


Prelude> applyTax 15
17.79


Prelude> applyTax (15 :: Int)

<interactive>:1:1:
    No instance for (Fractional Int)
      arising from a use of ‘applyTax’ at <interactive>:1:1-20
    Possible fix: add an instance declaration for (Fractional Int)
    In the expression: applyTax (15 :: Int)
    In the definition of ‘it’: it = applyTax (15 :: Int)
```

Haskell et les listes

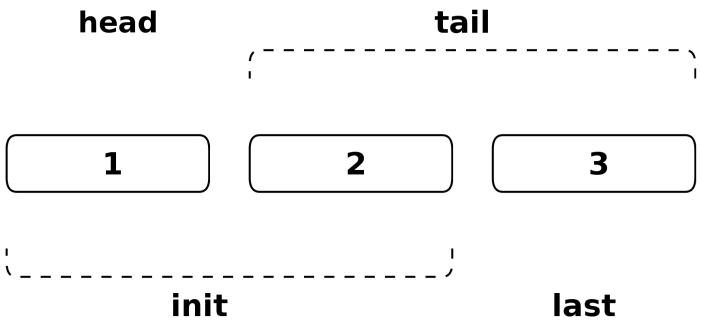
- [] : définit une liste

```
Prelude> [1,2,3]
[1,2,3]
```

- : : ajoute en élément en tête de liste

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

- Découpage d'une liste



Lecture d'un fichier

Ouverture du fichier

```
Prelude> fichier <- readFile "../Data/LesMiserables_T1.txt"
Prelude> :t fichier
fichier :: String
```

Lecture d'un fichier

Ouverture du fichier

```
Prelude> fichier <- readFile "../Data/LesMiserables_T1.txt"
Prelude> :t fichier
fichier :: String
```

Découpage en lignes

```
Prelude> :t lines
lines :: String -> [String]
```

Lecture d'un fichier

Ouverture du fichier

```
Prelude> fichier <- readFile "../Data/LesMiserables_T1.txt"
Prelude> :t fichier
fichier :: String
```

Découpage en lignes

```
Prelude> :t lines
lines :: String -> [String]


Prelude> let lignesMT1 = lines fichier
Prelude> take 3 lignesMT1
["The Project Gutenberg EBook of Les mis\233rables Tome I, by Victor Hugo","",
"This eBook is for the use of anyone anywhere at no cost and with"]
```

Lecture d'un fichier

Ouverture du fichier

```
Prelude> fichier <- readFile "../Data/LesMiserables_T1.txt"
Prelude> :t fichier
fichier :: String
```

Découpage en lignes

```
Prelude> :t lines
lines :: String -> [String]


Prelude> let lignesMT1 = lines fichier
Prelude> take 3 lignesMT1
["The Project Gutenberg EBook of Les mis\233rables Tome I, by Victor Hugo","",
"This eBook is for the use of anyone anywhere at no cost and with"]
```

Découpage en mots

```
Prelude>:t words
words :: String -> [String]

Prelude> let motsMT1 = words fichier
Prelude> take 10 motsMT1
["The","Project","Gutenberg","EBook","of","Les","mis\233rables","Tome","I,","by",
"Victor","Hugo","This","eBook","is","for","the","use","of","anyone"]
```

Suite

```
fichier <- readFile "LesMiserables_T1.txt"
let lignesMT1 = lines fichier
mapM_ print $ take 10 lignesMT1
words .  concat . (Data.List.intersperse " ") $ take 100 lig
(sans intersperse les fins de lignes sont collées au débuts
mieux :
let wordsNLignes n = words . concat . (Data.List.intersperse
wordsNLignes 15 lignesMT1
-- fonction motSuiv : permet de savoir le mot qui suit un au
-- malgré les sauts de ligne
:{
 let { motSuiv m [] = []
     ; motSuiv m [_] = []
     ; motSuiv m (x:xs) = if (x == m) then ((head xs):(motSu
     }
:}
motSuiv "Jean" $ wordsNLignes 4000 lignesMT1
```