



PYTHON ET LES EXPRESSIONS RÉGULIÈRES

QU'EST-CE QUE C'EST ?

C'est ...

Une **expression** ...

Un ensemble ou une suite de mots disposés dans un certain format

... régulière

que l'on retrouve en plusieurs exemplaires, se ressemblant plus ou moins.

MAIS VOYONS UN EXEMPLE TOUT DE SUITE

Considérons les lignes suivantes situées dans un fichier texte.

Numéro 19 place 889 etage 65

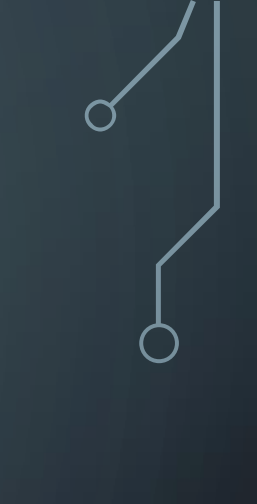

Atelier Duchamp, avenue Jean demont

00601 VilleTruc

Numéro 65 place 459 etage 45

Atelier Montu, avenue Ticotico

05103 Villehaha



En général, les expressions régulières sont utilisées lors de la lecture de fichiers pour extraire des informations en cherchant des « **pattern** » particuliers. Ce que j'appelle par un « **pattern** » c'est un ensemble de mots ou une suite de mots disposés d'une certaine manière (que l'on retrouve plusieurs fois dans le fichier lu ou non).

Si dans le fichier texte de la diapositive précédente, on veut extraire le code postal de chaque adresse, il faudra rechercher le pattern correspond à la ligne où se trouve le code postal puis de chercher dans la ligne l'information que l'on veut.

Mais voyons tout d'abord de quoi peut être constituée une ligne.



DE QUOI PEUT ÊTRE CONSTITUÉE UNE LIGNE ?

Une lettre (minuscule ou majuscule), un nombre, une ponctuation, un élément non lettre non nombre non ponctuation (`#@$*+~*/ ...`), un espace, une tabulation, un saut de ligne.

En programmation, nous allons utiliser différents symboles pour rechercher ces éléments.

Elément	Symbole	
fin de ligne	\$	
début de ligne	^	
n'importe quel élément	.	
un chiffre	[0-9] ou \d	
tout sauf un chiffre	\D	
une lettre minuscule	[a-z]	\w
une lettre majuscule	[A-Z]	
tout sauf une lettre	\W	
lettre(min/maj) ou chiffre	[a-zA-Z0-9]	
une tabulation	\t	
un espace	\s ou []	
tout sauf un espace ou une tabulation	\S	
saut de ligne	\n	

Explications

Pour plus de clarté, je ne traiterai pas les différents symboles dans l'ordre présenté dans le tableau.

.

Le « . » signifie n'importe quel élément, que ce soit une lettre, un chiffre, une ponctuation, ou un autre élément ne faisant pas parti d'une des catégories précédentes.

Exemples :

α , 2, ?, ,, [, =, +, ^, μ , \sim , etc.

[0-9] ou \d

Ces deux symboles représentent un chiffre (0, 1, 2, ..., 9). Rien de compliqué.

En ce qui concerne le \d, je ne peux rien vous dire dessus, c'est comme ça.

Pour le deuxième, [0-9], remarquez l'utilisation des crochets « [] ».

Mais parlons-en tout de suite !

LES CROCHETS []

Lorsque l'on cherche un certain élément, on « peut » le spécifier entre les crochets, ou non. Si par exemple on cherche le chiffre 1, on peut écrire : [1], ou 1 dans l'expression régulière. Au choix.

Dans le cas de tous les chiffres, [0-9], on précise qu'on recherche soit 0, soit 1, soit 2, ..., soit 9. On ne peut pas écrire 0-9 comme cela. Il faut mettre des crochets. [0-9] correspond également à [0123456789].

On peut également écrire [\d], mais cela n'a pas d'intérêt.

Attention ! On ne cherche qu'UN élément parmi tous les éléments situés entre les crochets !

Si vous voulez chercher le nombre 99, [0-9] ne prendra qu'un 9. Il faudra écrire [0-9][0-9] ou \d\d.

The background is a dark blue gradient. In the corners, there are decorative white line art elements resembling circuit boards or neural networks, with lines and small circles.

`\D`

Ce symbole indique que l'on veut TOUT sauf un chiffre. On cherche donc une lettre, une ponctuation, un élément non chiffre non lettre non ponctuation.

`[a-z]` ou `\w`

Une lettre qui est comprise dans l'alphabet latin (a, b, c, d, e, f, ..., z).

Lorsque l'on utilise `[a-z]`, la lettre doit être en minuscule !

Le deuxième symbole `\w` ne spécifie pas si l'on veut une minuscule ou une majuscule (a, b, c, ..., z mais aussi A, B, C, ..., Z).

[A-Z] ou \w

Même chose que précédemment, mais [A-Z] précise que l'on cherche une majuscule.

Le \w ne fait pas la différence entre minuscule et majuscule.

The background is a dark blue gradient. In the corners, there are decorative white line art elements resembling circuit boards or neural networks, with lines and small circles connecting them.

\W

Tout sauf une lettre (minuscule et/ou majuscule).

Les éléments qui seront acceptés sont un chiffre, une ponctuation, un élément non chiffre, non lettre, non ponctuation.

[a-zA-Z0-9]

Une lettre minuscule OU une lettre majuscule OU un chiffre. C'est tout ! Pas de ponctuation ou autre.

Notez que l'on peut également l'écrire de cette façon :

[\w\d]

`\s` ou `[]`

C'est un espace ! Et oui ! Les espaces aussi sont important dans les expressions régulières. Sinon, on ne pourrait pas différencier les mots.

Pouvez-vous me dire combien de mots composent cette phrase ?

Remarquez que dans cet exemple, il n'y a pas de séparateur entre chaque mot qui nous permettrait de mettre la phrase dans une liste par exemple (avec `split()`).

\t

Une tabulation. Vous savez la touche rectangle avec 2 flèches qui est située à gauche de la touche « a » de votre clavier ? Et bah c'est ça une tabulation.

Attention ! Ne pas confondre avec un espace ! Soit vous avez une tabulation soit un espace !

En fait, une tabulation c'est plusieurs espaces. Donc vous pouvez chercher la tabulation avec « \s » à condition de le répéter autant qu'il le faut.

The background is a dark blue gradient. In the corners, there are decorative white line art elements resembling circuit boards or neural networks, with lines and small circles.

`\S`

Tout sauf un espace ou une tabulation.

Une lettre, un chiffre, une ponctuation, ... mais pas d'espace ni de tabulation !

The background is a dark blue gradient. In the corners, there are decorative white line art elements resembling circuit boards or neural networks, with lines and small circles.

`\n`

Ce symbole doit vous paraître familier. En effet on l'utilise beaucoup pour sauter des lignes.

Dans les fichiers texte, il est présent à chaque fin de ligne.

^

Le début de ligne ! Très pratique dans le cas où vous cherchez un élément situé en début de ligne.

En effet s'il y a un chiffre qui vous souhaitez avoir et qu'il est situé en début de ligne vous écrirez ainsi :

« `^\d` » ou bien « `^[0-9]` »

Votre ligne commence donc forcément par un chiffre. Si c'est une lettre, remplacer par le symbole correspondant `\d` ou `[a-z]` ou `[A-Z]`.

AUTRE SIGNIFICATION DE ^

S'il est utilisé dans des crochets, par exemple comme cela : `[^0-9]`, cela signifie que l'on « ne veut pas » de chiffre. En effet le ^ permet d'exclure ce que l'on ne souhaite pas avoir. On pourrait aussi écrire `[^\d]`.

Attention ! Il exclu tout ce qui se trouve après lui dans les crochets.

`[^ABCD]` signifie que l'on ne veut ni A ni B ni C ni D.

\$

La fin de ligne. Même principe que pour le début de ligne. Si l'on cherche un élément située en fin de ligne, un chiffre par exemple, on écrirait :

`\d$` ou bien `[0-9]$`

Dans le cas d'une lettre, même principe.

`\w$` ou bien `[a-z]$` ou bien `[A-Z]$`

Vous avez là presque tous les outils pour manipuler les expressions régulières, mais il vous en manque quelques-uns.

Si par exemple, vous souhaitez chercher le mot « Bonjour » en début de ligne, comment écririez-vous le pattern correspondant ?

Comme ceci : `^\w\w\w\w\w\w\w`

Ou bien comme ceci : `^[A-Z][a-z][a-z][a-z][a-z][a-z][a-z]`

Avouez que ce n'est pas très pratique. Pour y remédier, on va utiliser des quantifieurs.

LES QUANTIFIEURS

Dans le cas où on aurait une suite de lettres, au lieu de réécrire X fois le symbole de la lettre « $\backslash d$ », on peut utiliser ce qu'on appelle un quantifieur. C'est un élément qui nous permet de spécifier le nombre de fois que l'on veut tel ou tel élément.

Symbole	Signification
*	De 0 à indéfini
+	De 1 à indéfini
?	1 ou 0

EXEMPLE

Reprenons notre « Bonjour » en début de ligne. Au lieu de répéter le `\d`, on peut écrire tout simplement l'un des 4 pattern suivants :

`^\d*`

`^[A-Z][a-z]*`

`^\d+`

`^[A-Z][a-z]+`

Avouez que c'est bien plus simple !

Attention tout de même ! Le quantifieur ne s'applique qu'à l'élément situé juste avant. Si vous écrivez : `\w\d+`, vous recherchez une lettre puis un ou plusieurs chiffres.

Vous avez presque tous les éléments qu'il vous faut mais il vous en manque un.

En effet, si vous remontez dans la diapositive qui parlait des crochets, je vous ai spécifié que vous pouviez rechercher le chiffre 1 soit en écrivant [1] ou 1. On s'intéresse ici au deuxième cas, lorsque le « 1 » n'est pas entouré de crochets. En effet, pour les lettres et les chiffres, vous pouvez les écrire tels que vous les cherchez sans mettre de crochets, ni utiliser les symboles.

En effet si vous souhaitez chercher « Bonjour » en début de ligne, vous pouvez écrire ^Bonjour, tout simplement. \w signifie une lettre, n'importe laquelle.

Si on reprend ^[A-Z][a-z][a-z][a-z][a-z][a-z][a-z], cela ne correspond pas forcément au mot « Bonjour ». Il peut correspondre à Jardine, Rebondi, Apprend, etc.

Si une ligne qui commence forcément par « Bonjour » en début de ligne, mieux vaut utiliser ^Bonjour plutôt que les autres. Sinon vous aurez d'autres lignes indésirables.



Tout cela pour vous dire qu'il y a des éléments dont vous devez ajouter un autre élément devant pour rechercher l'élément.

On appelle ces éléments, les métacaractères.



LES MÉTACARACTÈRES

Ou caractères spéciaux. Ce sont les suivants :

! ^ \$ () [] { } ? + * . \ |

En effet, ces caractères ont une certaine signification (remarquez le ^ et le \$ que je vous ai présenté précédemment). Vous ne pouvez pas les écrire comme je vous l'ai expliqué avec le « Bonjour » sans les crochets. De même, mettre un de ces caractères entre les crochets ne servira à rien.

Pour rechercher l'élément ^ par exemple, il faut l'échapper.

ÉCHAPPEMENT

Échapper un élément signifie rajouter un backslash, ou antislash « \ » devant cet élément.

Pour rechercher l'élément ^, vous devez donc écrire \^ . Tout simplement.

Comment allez vous faire pour rechercher l'élément « \ » dans ce cas ? Et bien comme ceci : \\

Mettez donc un « \ » (pas les guillemets) devant chaque métacaractère pour rechercher ce métacaractère en tant que caractère !

AUTRES METACARACTÈRES UTILES

Si vous regardez la liste des métacractères que je vous ai présenté, je ne vous en ai expliqué que quelques-uns. Parmi ceux listés, vous devez connaître : `^ $ [] ? + * . \`

En voici d'autres !

LES PARENTHÈSES ()

Lorsque vous recherchez une certaine information située dans une ligne, il serait très pratique de garder cette information dans une variable par exemple. C'est très bien de savoir comment chercher une donnée. Mais il serait encore mieux de pouvoir la garder et de l'utiliser. Pour cela, on utilise les parenthèses (). Il suffit juste de mettre dedans ce que vous souhaitez garder. On parle de capture.

EXEMPLE

Soit la ligne suivante :

Pierre 21 etudiant Paris

Vous souhaitez garder l'âge de Pierre en mémoire mais comment faire ? Il suffit de rechercher le pattern correspondant à cette ligne en y ajoutant des parenthèses autour de l'âge comme suit :

$$^{\wedge}\backslash w^{*}\backslash s(\backslash d^{*})\backslash s$$

Il suffit ensuite d'affecter tout cela dans une variable avec une fonction contenue dans un module permettant de manipuler les expressions régulières que l'on verra plus tard.

Remarque : si vous observez le pattern présenté, notez que je m'arrête à l'espace situé juste après l'âge. En effet, il n'est pas nécessaire d'aller plus loin puisque j'ai spécifié que mon pattern est situé en début de ligne avec le $^{\wedge}$. Dans le cas où vous ne spécifiez pas le début de ligne, il faut écrire le pattern correspondant à la ligne entière puisqu'il se pourrait que le pattern que je viens d'écrire (sans le $^{\wedge}$) pourrait se trouver autre part dans le fichier. Vous pouvez également commencer par la fin en le spécifiant par $\$$ comme ceci :

$$\backslash s(\backslash d^{*})\backslash s\backslash w^{*}\$$$

En raboutant les deux patterns, vous obtenez le pattern pour la ligne entière en enlevant le $\backslash s$ à la fin du premier pattern ou le $^{\wedge}$ au début du premier pattern.

EXPRESSION

Il existe une deuxième utilité des parenthèses. Outre sa fonction de capture, il a aussi une fonction d'expression.

Comment pouvez-vous créer un pattern qui recherche la suite de bonjour suivante :

« bonjourbonjourbonjour »

En entourant « bonjour » avec des parenthèses ! Suivez l'exemple :

(bonjour)* ou bien (bonjour)+

Rappelez-vous qu'avec les crochets, on ne cherche qu'un élément parmi tout ce qu'il y a dans les crochets. Avec [bonjour], on cherche soit la lettre « b », soit « o », soit « n », soit « j », soit « r », soit « u ».

LE PIPE « | »

Si vous vous souvenez parmi les opérateurs logiques utilisés en C, le `||` signifie « ou ». C'est pareil ici, mais on en a qu'un seul. Si dans notre pattern, on recherche telle ou telle chose on peut le spécifier par le `|`.

Si vous recherchez la lettre « a » ou la lettre « b » écrivez comme suit : `[a|b]` ou alors `a|b` plus simplement. Ce pattern va reconnaître soit la lettre « a » soit la lettre « b ».

Par contre si vous utilisez `\w|\w`, et bien cela ne servira pas à grand-chose. Entre une lettre et une lettre, le choix n'est pas si compliqué. Par contre vous pouvez le faire pour un chiffre et une lettre : `\d|\w`

COMBINAISON AVEC LES PARENTHÈSES ()

On peut utiliser le « | » avec les « () » pour rechercher tel ou tel expression. Suivez l'exemple :

(bonjour | salut)

Dans ce cas, on recherche soit « bonjour », soit « salut ».

LES ACCOLADES {}

Agit comme un quantifieur. Placé après une expression ou un élément.

$\{n\}$: on recherche exactement n fois l'élément situé avant ou l'expression.

$\{n, m\}$: on recherche n à m fois

$\{n, \}$: au moins n fois

$\{,m\}$: au maximum m fois

EXEMPLES

$A\{4\}$: on recherche exactement AAAA

$B\{3, 5\}$: on recherche BBB ou BBBB ou BBBBB

$(\text{truc1 2})\{2\}$: on recherche au moins truc1 2truc1 2 (mais aussi truc1 2tuc1 2truc1 2, ...)



$6\{,3\}$: on recherche 6 ou 66 ou 666

À votre avis que recherche t-on avec : $\text{ceci}\{3\}$?



Je vous laisse le loisir de vous informer sur les métacaractères que je ne vous ai pas expliqué.

Vous savez maintenant comment écrire un pattern. Pour les manipuler proprement, on va maintenant utiliser des fonctions propres aux expressions régulières.



MODULE DES EXPRESSIONS RÉGULIÈRES

Les fonctions qui permettent de manipuler les expressions régulières sont situées dans le module `re`.

Pour les utiliser il suffit d'importer cette bibliothèque comme suit :

```
import re
```

N'oubliez pas qu'il existe plusieurs façons d'importer un module. Dans ce cas ci, spécifiez le nom du module (`re`) devant chaque fonction que vous allez utiliser provenant de ce module.

RECHERCHE DE MOTIF (PATTERN)

La fonction **search()** permet de rechercher un motif (pattern) au sein d'une chaîne de caractères avec la syntaxe suivante :

search(motif, chaîne)

Si le motif est trouvé, la fonction renvoie une instance d'un objet :

<_sre.SRE_Match object at >

Sinon renvoie None. On peut faire précéder la fonction d'un « if » dans le cas d'un test.

EXEMPLE

```
import re

chaine = "salut wesh bonjour coucou"

print re.search('wesh', chaine)
print re.search('hi', chaine)

if re.search('hello', chaine):
    print "C'est dedans"

else:
    print "C'est pas dedans"
```

```
<_sre.SRE_Match object at 0x7f9682cb44a8>
None
C'est pas dedans
```

COMPILATION D'EXPRESSION RÉGULIÈRE

Il est pratique de compiler préalablement compiler l'expression régulière à l'aide de la fonction **compile()** qui renvoie un objet de type expression régulière :

<_sre.SRE_Pattern object at >

Insérez l'expression régulière à compiler dans les parenthèses, et l'entourer de « » ou de ' '.

EXAMPLE

```
import re

motif = re.compile("^Bonjour")

print motif

chaine = "Bonjour toi"

print motif.search(chaine)

chaine2 = "Hey toi ! Bonjour !"

print motif.search(chaine2)
```

```
<_sre.SRE_Pattern object at 0x7fdb7f9c3f50>
<_sre.SRE_Match object at 0x7fdb7f886510>
None
```

GROUPE

Lorsque l'on capture plusieurs éléments dans un pattern (avec les parenthèses), on crée des groupes de capture. Pour accéder à chaque « groupe » (chaque paire de parenthèses), on utilise la fonction **group()** sur la variable qui contient les éléments capturés.

group(0) donne la totalité de la correspondance

group(1) donne le premier élément

group(2) donne le deuxième

Etc.

On distingue aussi les fonctions **start()** et **end()** qui donnent le début et la fin de la zone qui correspond à l'expression régulière. À utiliser de la même manière que la fonction **group()**. Ne prend par contre pas d'arguments.

EXEMPLE

```
import re

motif = re.compile('\s(\d*)\s.*\s(\d*)')

resultat = motif.search("J'ai eu 15 et toi 17")

print resultat.group(0)
print resultat.group(1)
print resultat.group(2)
print resultat.start()
print resultat.end()
```

```
15 et toi 17
15
17
7
20
```

FONCTION FINDALL()

Si dans une ligne, votre pattern est trouvé plusieurs fois, vous pouvez, avec la fonction **findall()** récupérer toutes les correspondances trouvées avec votre pattern.

Renvoie une liste des éléments trouvés.

EXEMPLE

```
import re

motif = re.compile('\d*\.\d*')

resultat = motif.findall("J'ai eu 15.6 et toi 17.9")

print resultat

motif = re.compile('(\d*)\\.(\d*)')

resultat = motif.findall("J'ai eu 15.6 et toi 17.9")

print resultat
```

```
['15.6', '17.9']
[('15', '6'), ('17', '9')]
```

REPLACEMENTS

Il est possible de remplacer dans une chaîne de caractère un élément par un autre avec la fonction **sub()**.

sub(chaine1, chaine2) remplace toutes les occurrences trouvées par l'expression régulière dans chaine2 par chaîne1.

Si on ne souhaite remplacer que les n premières occurrences trouvées dans chaine2, on peut le préciser avec **count** (count=n).

EXEMPLE

```
import re

motif = re.compile('\d\d')

print motif.sub("6", "J'ai eu 15 et toi 17")
print motif.sub("6", "J'ai eu 15 et toi 17", count=1)
```

```
J'ai eu 6 et toi 6
J'ai eu 6 et toi 17
```

EXERCICES POUR VOUS EXERCER !!!

1) Comment écririez-vous l'expression régulière pour rechercher la ligne suivante :

START NB 12.56.7 silhouette marche.49 fin

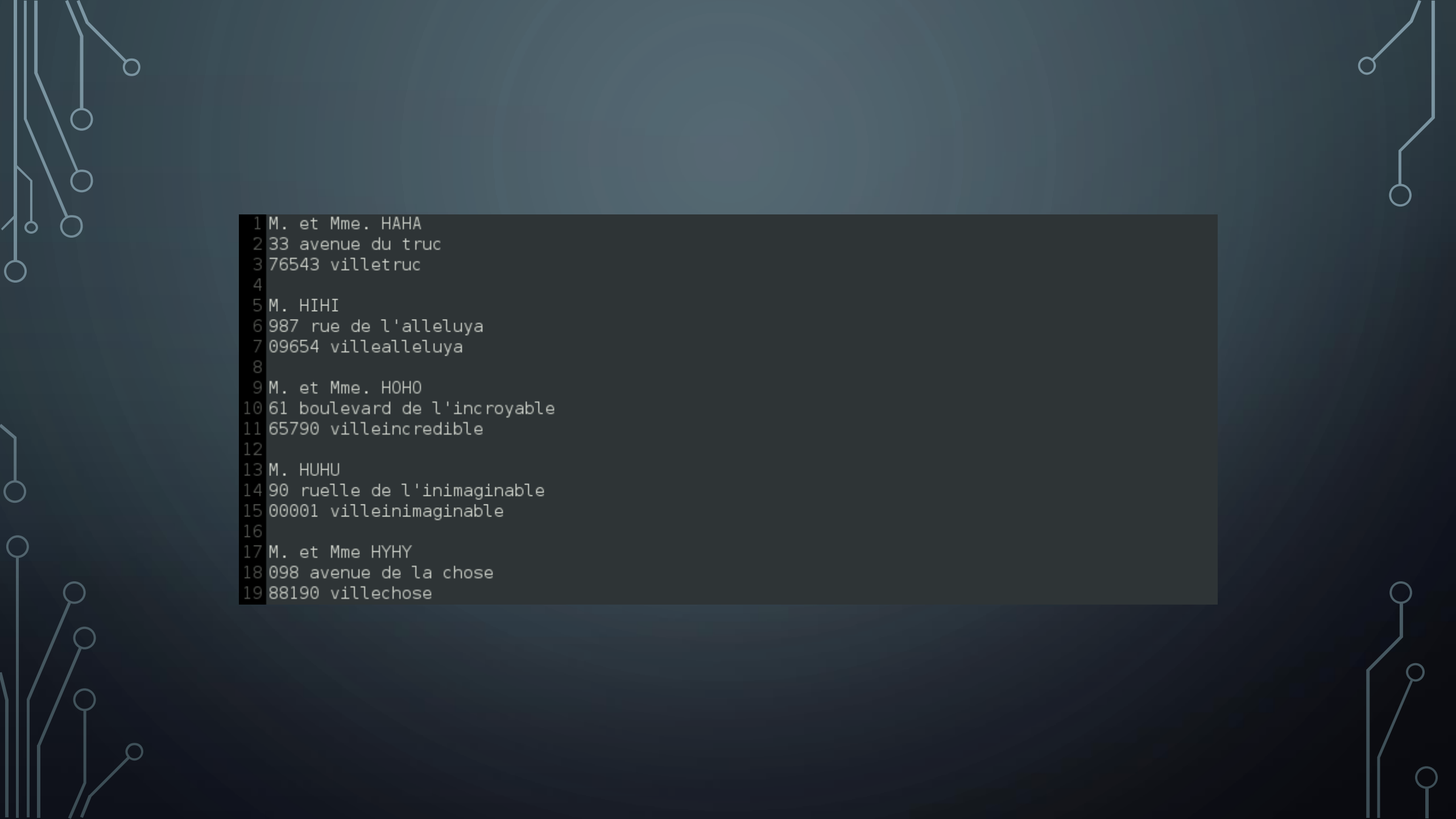
2) Reprenez ce que vous venez d'écrire mais maintenant gardez le « fin » et affichez la position de début et fin de la correspondance.

3) Soit le fichier texte dans la diapositive suivante : y est répertorié les noms de chaque famille et leur adresse.

Écrire un programme qui lit le fichier et récupère :

- Le nom de famille de chaque famille
- Le numéro de leur adresse
- Le nom de la rue dans laquelle chacun habite
- Leur code postal
- La ville

Vous êtes libre de choisir le format dans lequel vous stockerez vos données.



```
1 M. et Mme. HAHA
2 33 avenue du truc
3 76543 villetruc
4
5 M. HIHI
6 987 rue de l'alleluya
7 09654 villealleluya
8
9 M. et Mme. H0H0
10 61 boulevard de l'incroyable
11 65790 villeincredible
12
13 M. HUH0
14 90 ruelle de l'inimaginable
15 00001 villeinimaginable
16
17 M. et Mme HYHY
18 098 avenue de la chose
19 88190 villechose
```