

Avrile Floro (n° étudiant : 22000086)

Algorithmique et structures de données 1

Le 27 décembre 2023



Table des matières

1	PM 11	3
1.1	Le fichier PM_11.lisp	3
1.1.1	La fonction Eliza	3
1.1.2	La fonction read-input et split-by-one-space	4
1.1.3	La fonction match	5
1.1.4	La fonction put	7
1.1.5	La fonction \$	7
1.1.6	La fonction answer	7
1.1.7	La fonction imprimer-liste	8
1.1.8	La fonction print-output	9
1.1.9	La fonction print-with-joker	9
1.1.10	La fonction butfirst	10
1.1.11	La fonction +z	10
1.1.12	La fonction appartient	11
1.1.13	La fonction nompropre	11
1.2	Le fichier PM_11_data.lisp	12
1.3	Utilisation du programme	14
1.3.1	Un échange fictif	14
1.3.2	Les limitations de notre pattern-matcher	19
Annexes		21
.1	L'intégralité du fichier "PM_11.lisp"	21
.2	L'intégralité du fichier "PM_11_data.lisp"	26

Algorithmique et structures de données 1

PM 11

Avrile Floro

Étudiante n°22000086

1 PM 11

H. Exercice

Écrire un pattern-matcher simple de type Eliza ; l'idée est de simuler une interview par un utilisateur (supposez que votre programme soit en ligne sur un site web) d'une femme ou d'un homme politique en campagne. Choisissez de préférence un politique ayant beaucoup de stéréotypes (formules figées) caractéristiques, et inspirez-vous de ses discours et de ses interviews. Arrangez-vous pour que le dialogue puisse durer au moins vingt échanges. Le politique peut poser des questions aussi. Une variante amusante pourrait se faire avec des SMS, des smileys, etc.

Envoyez le pattern-matcher par mail avec le sujet « Exo PM 11 » (code et données compressées + PDF de commentaires sur vos difficultés).

Pour cet exercice, nous avons suivi les recommandations proposées par le cours et nous avons créé deux fichiers. Le premier `PM_11.lisp` contient le moteur du pattern-matcher alors que le second fichier `PM_11_data.lisp` contient les variables globales et notamment les listes de propriété.

L'ensemble des fichiers sont disponibles intégralement en annexes.

1.1 Le fichier `PM_11.lisp`

Nous allons commencer par présenter le fichier `PM_11.lisp` qui est le fichier principal de notre pattern-matcher.

Tout d'abord, nous commençons par importer le fichier `PM_11_data.lisp`, car sur les recommandations du cours, nous avons décidé de modulariser notre programme afin de conserver dans un autre fichier les données.

```
1 ; Nom ..... : PM_11.lisp
2 ; Rôle ..... : pattern matcher en lisp
3 ; Auteur ..... : Avrië Floro
4 ; Version ..... : V0.1 du 14/12/23
5 ; Licence ..... : réalisé dans le cadre du cours d'Algo 1
6 ; Usage : lisp puis (load "PM_11.lisp") puis (Eliza patterns)
7
8 (load "PM_11_data.lisp")
9 ; séparation du fichiers pour les données
```

1.1.1 La fonction `Eliza`

La fonction `Eliza` est la fonction principale de notre pattern matcher et celle qu'il convient d'appeler pour lancer le programme.

Nous avons apporté quelques modifications à la fonction `Eliza` proposée par le cours. Notamment, nous avons réinitialisé la valeur de `Variables` et de `Resultats` lors de chaque itération du loop. `Variables` permet de stocker les variables du pattern-matcher et `Resultats` permet de stocker les résultats.

Par ailleurs, nous avons ajouté un `princ "Utilisateur: "` afin de faciliter le suivi du dialogue entre les interlocuteurs. Pour la même raison, nous avons ajouté deux (`terpri`) permettant des sauts de ligne. Cela rend le dialogue plus clair et lisible.

```
11 ; NAME : Eliza > fonction principale du pattern matcher
12 ; ARGS : patterns
13 ; USAGES : (Eliza patterns)
14 ; GLOBALS : Variables, Resultats, patterns
15 ; CALL : answer, read-input, match
16 ; USER : top level
17
18 (defun Eliza (patterns)
19   (loop
20     (setq Variables nil) ; réinitialisation de Variables
21     (setq Resultats nil) ; réinitialisation de résultats
22     (princ "Utilisateur: "); ajout d'un saut de ligne
23     (or (answer (cdr (assoc (read-input) patterns :test #'match))))
24     (return))
25   (terpri)
26   (terpri) ) )
```

1.1.2 La fonction `read-input` et `split-by-one-space`

La fonction `read-input` lit une entrée jusqu'à un retour-charriot et renvoie la liste de ce qu'elle a lu. Afin de pouvoir gérer les entrées utilisateurs contenant plusieurs mots, nous avons utilisé la fonction `read-line` qui lit la ligne entière. Néanmoins, cette fonction renvoie l'intégralité de la ligne comme une seule string. Nous avons donc dû utiliser la fonction `split-by-one-space` qui permet de découper une string en plusieurs selon un caractère délimiteur, en l'espèce l'espace.

Nous avons utilisé la fonction `mapcar` avec `#'intern`, cela nous permet d'obtenir comme résultat une liste dont chacun des mots est un symbole.

```
28 ; NAME : read-input ; lit ligne, divise en mots, convertit mots en symb
29 ; ARGS : none
30 ; USAGES : (read-input) > bonjour au revoir > (bonjour au revoir)
31 ; GLOBALS : none
32 ; CALL : split-by-one-space
33 ; USER : Eliza
34
35 (defun read-input ()
36   (mapcar #'intern (split-by-one-space (read-line))))
37 ; séparation mots aux espaces, transformation en symbole et ajout lst
```

La fonction `split-by-one-space` est proposée par le très utile `lispcookbook` (source : <https://lispcookbook.github.io/cl-cookbook/strings.html>). Elle permet de séparer une chaîne de caractères selon un délimiteur, qui est ici l'espace. On obtient en sortie plusieurs chaînes de caractères distinctes.

```
40 ; NAME : split-by-one-space > divise une str en mots grâce aux espaces
41 ; ARGS : string
42 ; USAGES : (split-by-one-space) > (split-by-one-space "un essai") ("un" "essai")
43 ; GLOBALS : none
44 ; CALL : none
45 ; USER : read-input
46 ; SOURCE : https://lispcookbook.github.io/cl-cookbook/strings.html
47
```

```

48 * (defun split-by-one-space (string)
49   (loop for i = 0 then (1+ j)
50     as j = (position #\Space string :start i)
51     collect (subseq string i j)
52     while j ) )

```

Nous avons eu quelques difficultés initiales afin de trouver les bonnes combinaisons. Nous avons tout d'abord réalisé qu'il convenait d'utiliser des symboles et donc d'avoir recours à `intern`. En outre, nous pensions initialement utilisé `read` mais cela ne convenait pas pour les entrées à plusieurs mots. Par ailleurs, nous connaissions déjà la fonction `split-by-one-space` qui est très utile afin de bénéficier de cette fonctionnalité de découpage des strings sans bibliothèque supplémentaire.

1.1.3 La fonction `match`

La fonction `match` est une fonction centrale de notre pattern-matcher puisqu'elle permet d'associer les entrées utilisateurs et les patrons.

Nous avons corrigé certaines des erreurs du code fourni dans le cours (des parenthèses manquantes ou supplémentaires autour de `cdr`, etc). Par ailleurs, nous avons adapté les clauses à notre cas d'utilisation.

Bien entendu, nous avons conservé les clauses basiques. C'est-à-dire que si l'entrée utilisateur et le patron sont vides, l'association est réussie.

Nous avons rajouté une clause permettant de vérifier, en cas d'entrée utilisateur vide avec un patron correspondant à `*`, que le `cdr` du patron est nul. Nous avons été confrontée à ce bug, qui ne parcourait pas convenablement la fin du patron dès lors qu'un `*` était utilisé. Cela conduisait à des associations erronées.

Une fois que nous avons vérifié que notre entrée vide n'est pas associée à un ultime caractère `*`, nous pouvons renvoyer `nil` pour tous les autres cas dans lesquels l'entrée ou le patron seraient vides sans que l'autre ne le soit également.

```

54 ; NAME : match > associe les entrées et les patrons
55 ; ARGS : input-form, pattern
56 ; GLOBALS : none
57 ; CALL : $ et appartient (via apply)
58 ; USER : Eliza
59
60 (defun match (input-form pattern)
61   ;(print (list 'input-form input-form 'pattern pattern))
62   (cond
63     ((and (not input-form) (not pattern))) ; entrée & patron finis : réussite
64
65     ((and (not input-form) (eq (car pattern) '*)) ; entrée finie et patron = *
66      (not (cdr pattern))) ; vérifie si patron fini après '*'
67
68     ((not input-form) nil) ; entrée finie mais pas patron > nil
69
70     ((not pattern) nil) ; patron vide mais pas entrée > nil

```

Lorsque le car du patron est une liste

Nous testons ensuite si le car du patron est une liste non nulle (avec `consp`). Nous avons ensuite introduit un second `cond` nous permettant de tester les différentes hypothèses dans lesquelles nous avons utilisé une liste.

Notre premier test consiste à vérifier si la liste est d'une longueur supérieure à 1 et si elle commence par `$`. Nous savons que nous sommes alors dans le cas de `($ A)` ou `($ A nompropre)` (par exemple).

Au sein de notre code, nous avons remplacé les ? proposés dans le cours par \$ afin de pouvoir facilement gérer les questions de l'utilisateur.

Si la condition est vérifiée, on applique la fonction \$ à ses arguments, à savoir le **car** de l'entrée utilisateur et le **cdr** du patron (s'il existe). On continue ensuite l'association.

Cela nous permet à la fois de gérer la récupération de plusieurs mots (car il peut y avoir une variable A et B et C, etc) et d'appliquer, quand nécessaire, des tests et notamment de vérifier si un mot est un nom propre.

```
72 ((consp (car pattern)) ; si le 1er elt est un cons (une liste non vide)
73   (cond
74     ((and ; c'est ($) suivi de quelque chose d'autre dans la lst ($ A __)
75       (and (> (list-length (car pattern)) 1) (eq (caar pattern) '$))
76       ; applique la fonction $ et passe les arguments
77       (apply (caar pattern) (cons (car input-form) (cдар pattern)))
78       ; on continue
79       (match (cdr input-form) (cdr pattern)) ) ) )
```

La seconde possibilité de notre sous-cond correspond au cas où la liste ne contient qu'un élément et qu'il s'agit de \$. Dans ce cas-là, nous n'avons pas besoin d'utiliser **apply**. Nous appelons simplement la fonction **put** et nous y ajoutons notre **joker**. Nous continuons ensuite l'association.

```
81 ((and ; quand c'est juste ($)
82   (and (= (list-length (car pattern)) 1) (eq (caar pattern) '$)) ; longueur liste 1 et $
83   (put 'Variables 'joker (car input-form)) ; stocke dans variable l'entrée associée à $
84   (match (cdr input-form) (cdr pattern)) ) )
```

Finalement, la dernière possibilité lorsque le **car** du patron commence par une liste non nulle est qu'il s'agisse de la fonction **appartient**.

On vérifie si le premier élément de la liste est **appartient**. On appelle ensuite la fonction \$ (notamment afin de mettre **Variables** à jour). Finalement, on applique la fonction **appartient** à ses arguments (que l'on a modifiés en conséquence). On continue ensuite l'association.

```
86 ((and ; quand c'est la fonction appartient
87   (eq (caar pattern) 'appartient)
88   ($ (car input-form) (cadar pattern)) ; on appelle la fonction $
89   (apply (caar pattern) (cons (car input-form) (cddar pattern)))
90   ; on continue
91   (match (cdr input-form) (cdr pattern)) ) ) )
```

Conclusion (lorsque le car du patron est une liste) : Les difficultés principales que nous avons rencontrées lorsque le **car** du patron est une liste était la bonne gestion des différents cas. En outre, il a fallu modifier en conséquence les arguments à passer à **apply** selon notre cas d'espèce. Nous avons également réalisé qu'il convenait de mettre à jour **Variables** directement ou via un appel à la fonction \$.

La possibilité suivante de notre fonction **match** est tirée du cours à l'identique et correspond au traitement du joker * qui peut correspondre à une suite quelconque de mots, y compris rien. Cette partie est tirée du cours.

```
93 ((eq (car pattern) '*) ; si patron commence par *
94   (or (match input-form (cdr pattern)) ; OU
95       ; le mot suivant du patron est le 1er mot de l'entrée
96       (match (cdr input-form) pattern) ) )
97   ; on continue en gardant l'étoile
```

Finalement, la dernière possibilité traitée par notre fonction **match** est également tirée du cours à l'identique. Lorsque le premier élément de la forme d'entrée et celui du patron sont identiques alors on continue.

```

99 ((eql (car input-form) (car pattern)) ; 1er élt patron = 1er élt entrée
100 (match (cdr input-form) (cdr pattern)) ) ) ; on continue avec l'élt suivant

```

1.1.4 La fonction put

La fonction `put` est tirée à l'identique du cours.

Au cours de la création du programme, nous lui avons ajouté un `print` pour faciliter le suivi du programme. Nous l'avons supprimé dans sa version finale.

```

103 ; NAME : put ; met des valeurs dans les propriétés
104 ; ARGS : symbole, attribut, valeur
105 ; USAGES : (put 'Variables 'joker (car input-form))
106 ; GLOBALS : Variables
107 ; CALL : none
108 ; USER : match, $
109
110 (defun put (symbole attribut valeur) (setf (get symbole attribut) valeur) valeur)

```

1.1.5 La fonction \$

La fonction `$` est tirée à l'identique du cours, nous avons simplement modifié son nom, la renommant `$` plutôt que `?`.

```

113 ; NAME : $ ; applique les tests à input-word et mäj de Variables / si non nil
114 ; ARGS : input-word variable &optional test
115 ; GLOBALS : Variables
116 ; CALL : none
117 ; USER : match
118
119 (defun $ (input-word variable &optional test)
120 (and
121 (or (not test) (apply test (list input-word)))
122 ; si test n'est pas nil, on l'applique à input-word
123 (or (not variable) (put 'Variables variable input-word)) ) )
124 ; si variable n'est pas nil, on mäj Variables

```

1.1.6 La fonction answer

La fonction `answer` construit la réponse avec le patron de sortie trouvé.

Nous avons considérablement modifié la fonction `answer` proposée par le cours. Tout d'abord, nous avons géré la gestion des entrées utilisateurs qui ne sont pas comprises par le programme directement au sein des données (nous y reviendrons lors de l'explication du fichier `PM_11_data.lisp`).

En outre, nous avons modifié le processus d'impression afin de ne plus utiliser des `print` (ce qui était le cas dans le programme initial) mais une liste `Resultats` à la place. Cela nous permet notamment une impression sur une seule ligne et avec une présentation correcte.

En cas de correspondance, la fonction `answer` appelle la fonction `print-output`. Ensuite, elle inverse la liste `Resultats` obtenue et ajoute Fabien Roussel: en début de ligne pour faciliter la compréhension.

En outre, on gère la sortie du pattern-matcher avec l'impression d'un `nil` permettant de sortir du pattern-matcher en fin de conversation.

Nous avons eu quelques difficultés lorsque nous avons voulu modifier l'affichage des résultats afin d'utiliser une liste plutôt que des `print` (comme c'était le cas initialement). Néanmoins, le passage sous forme de liste nous a permis d'obtenir une présentation beaucoup plus satisfaisante. En outre, nous avons également eu quelques petites difficultés à mettre fin à la conversation grâce à `nil`. Nous avons dû rajouter manuellement ce `nil` sinon la conversation ne se terminait pas.

```
127 ; NAME : answer > gestion réponse utilisateur
128 ; ARGS : output-pattern
129 ; GLOBALS : Resultats
130 ; CALL : print-output, imprimer-liste
131 ; USER : Eliza
132
133 (defun answer (output-pattern)
134   (cond
135     ((and ; si correspondance
136          (print-output output-pattern) ; appel print-output
137          (setq Resultats (reverse Resultats)) ; inversion de la liste résultats obtenue
138          (push "Fabien Roussel:" Resultats)
139          (imprimer-liste Resultats) ) )
140
141     (t (setq Resultats (reverse Resultats)) ; pour terminer la conversation
142        (push "Fabien Roussel:" Resultats)
143        (imprimer-liste Resultats)
144        nil ) ) ) ; impression du nil pour sortir
```

1.1.7 La fonction `imprimer-liste`

Dans le cadre de notre aménagement des modalités d'impression du résultat, nous avons dû créer une fonction `imprimer-liste` qui permet l'impression de la liste d'une façon acceptable sur le terminal (c'est-à-dire sans guillemets ni retour à la ligne après chaque impression). Nous avons donc choisi d'utiliser `princ`. Après l'impression successive de chaque `car` de la liste, on ajoute un espace (car ils ont été supprimés lors du traitement initiale de l'entrée par la fonction `read-line` et `split-by-one-space`).

En particulier, nous avons dû forcer la terminaison de la fonction par un `t`, sinon la fonction renvoyait un `nil` qui avait pour résultat de mettre fin au programme.

```
1 ; NAME : imprimer-liste > imprime les éléments d'une liste
2 ; ARGS : liste
3 ; USAGES : (imprimer-liste liste) > (imprimer-liste '("bonjour" "je")) > bonjour je
4 ; GLOBALS : none
5 ; CALL : none
6 ; USER : answer
7
8 (defun imprimer-liste (liste)
9   (cond
10     ((not liste) nil) ; si la liste est finie, on renvoie nil
11     (t (princ (car liste)) ; sinon on princ le 1er élt de la lst
12        (princ " ") ; on ajoute un espace
13        (imprimer-liste (cdr liste)) ; on récurse
14        t ) ) ) ; renvoie t pour ne pas avoir un nil qui met fin au programme
```

1.1.8 La fonction print-output

La fonction `print-output` a été tirée du cours. Nous avons simplement corrigé la fonction en retirant les parenthèses inutiles autour du `cdr` de `output-pattern`.

```
163 ; NAME : print-output > la gestion des impressions en sortie
164 ; ARGS : output-pattern
165 ; USAGES : (imprimer-liste liste) > (imprimer-liste '("bonjour" "je")) > bonjour je
166 ; GLOBALS : none
167 ; CALL : none
168 ; USER : answer
169
170 (defun print-output (output-pattern)
171   (cond
172     ((atom output-pattern) (not output-pattern))
173     (t
174      (print-with-joker (car output-pattern))
175      (print-output (cdr output-pattern)) ) ) ) ; retrait () autour de cdr
```

1.1.9 La fonction print-with-joker

Nous avons considérablement modifié la fonction `print-with-joker` proposée par le cours et notamment afin d'accomoder le stockage des résultats au sein d'une liste (ce que nous effectuons dans notre version de la fonction). En outre, nous avons implanté la gestion des différents jokers pouvant être utilisés. Ainsi, nous avons également décidé d'utiliser `cond` plutôt qu'un `if` (comme dans la version initiale).

La première possibilité de notre `cond` consiste en la présence d'une liste dans la sortie. Dans ce cas, on vérifie si le premier élément de la liste est `>`, si c'est le cas on appelle alors la fonction `+z` (qui est la seule fonction utilisée pour les sorties) en lui passant l'argument qu'il convient.

```
179 ; NAME : print-with-joker > gestion de l'impression des jokers
180 ; ARGS : output-word &optional result
181 ; GLOBALS : Variables, Resultats
182 ; CALL : butfirst
183 ; USER : print-output
184
185 (defun print-with-joker (output-word &optional result)
186   (setq result ; maj de result selon le retour de cond
187     (cond ; cond plutôt que if
188       ; si sortie est une liste non nulle qui commence par >
189       ((and (consp output-word) (eq (car output-word) '>))
190        (+z (get 'Variables (cadr output-word))) ) ; on appelle +z
```

Une autre possibilité est la présence d'un joker `$` simple. Dans les faits, nous n'avons utilisé cette possibilité qu'une fois dans le cadre de notre `pattern-matcher` afin d'illustrer son fonctionnement mais elle offre moins de flexibilité que lorsqu'on utilise des fonctions supplémentaires (notamment `appartient`). Son utilité est donc assez faible.

```
193 ; si la sortie est $
194 ((and (eq output-word '$) (= (length (string output-word)) 1))
195  (get 'Variables 'joker) ) ; on récupère joker
```

La troisième possibilité est la présence d'une variable qui commence par > suivi de son identifiant. Nous avons utilisé cette possibilité à plusieurs reprises car elle permet l'utilisation conjointe de multiples variables, permettant un plus grand degré de personnalisation.

```
197 ; si la sortie est >var
198 ((eq (char (string output-word) 0) '#\>)
199 (get 'Variables (butfirst output-word)) )
```

Finalement, si aucune des trois options ne s'applique à la sortie, on la renvoie telle quelle. En outre, on associe le résultat obtenu en sortie du cond à la variable **result** que l'on ajoute ensuite à notre liste de résultats **Resultats**.

```
201 (t output-word) ) ) ; sinon renvoie la sortie
202 (push result Resultats) ) ; on push result dans lst Resultats
```

1.1.10 La fonction butfirst

La fonction **butfirst**, que nous devons écrire, renvoie un atome dont le nom est le premier argument moins le premier caractère. En outre, la fonction **intern** le résultat afin de créer un nouveau symbole. L'utilisation de **intern** est obligatoire au bon fonctionnement de notre programme.

```
206 ; NAME : butfirst
207 ; > renvoie un atome dont le nom est celui de l'arg moins première caractère
208 ; ARGS : mot
209 ; GLOBALS : none
210 ; CALL : none
211 ; USER : print-with-joker
212
213 (defun butfirst (mot)
214 (intern (subseq (string mot) 1)) )
215 ; isole la fin du mot à partir de 1 puis intern
```

1.1.11 La fonction +z

La fonction **+z** renvoie un mot avec un **z** à la fin. Le résultat est renvoyé sous forme de chaîne de caractères.

```
218 ; NAME : +z
219 ; > renvoie un mot avec un z à la fin
220 ; ARGS : mot
221 ; GLOBALS : none
222 ; CALL : none
223 ; USER : print-with-joker
224
225 (defun +z (mot)
226 (concatenate 'string (string mot) "z") )
227 ; ajoute z à la fin du mot, forme une str
```

1.1.12 La fonction appartient

La fonction `appartient` permet de tester si un mot appartient à une liste de mots. La fonction renvoie le mot s'il appartient effectivement à la liste ou `nil` sinon.

```
230 ; NAME : appartient
231 ;      > renvoie le mot s'il appartient à une liste donnée ou nil
232 ; ARGS : mot &rest lesoptions
233 ; GLOBALS : none
234 ; CALL : none
235 ; USER : match
236
237 (defun appartient (mot &rest lesoptions)
238   (car (member mot lesoptions)) )
239 ; vérifie si le mot appartient à la lst des lesoptions
```

1.1.13 La fonction nompropre

La fonction `nompropre` permet de tester si un mot est un nom propre, c'est-à-dire s'il commence par une majuscule. Si c'est le cas, la fonction renvoie `t`. En revanche, si le mot n'est pas un nom propre, alors la fonction renvoie `nil`.

```
242 ; NAME : nompropre
243 ;      > renvoie nil ou t selon que le mot commence par une maj
244 ; ARGS : mot
245 ; GLOBALS : none
246 ; CALL : none
247 ; USER : match
248
249 (defun nompropre (mot)
250   (upper-case-p (char (string mot) 0)) )
251 ; vérifie si le mot commence par une majuscule
```

1.2 Le fichier PM_11_data.lisp

Comme cela était recommandé dans les consignes de l'exercice, nous avons placé les définitions des variables globales et en particulier **Variables** dans un fichier séparé.

Le fichier est disponible intégralement en annexe.

La fonction personnalisée que nous avons trouvée la plus efficace et versatile a été la fonction **appartient**, qui est de loin celle que nous avons le plus utilisée.

Nous avons utilisé l'ensemble des fonctions personnalisées et des jokers que nous avons décrits précédemment.

```
1 ; Nom ..... : PM_11_data.lisp
2 ; Rôle ..... : données pour le pattern matcher
3 ; Auteur ..... : Avrile Floro
4 ; Version ..... : V0.1 du 14/12/23
5 ; Licence ..... : réalisé dans le cadre du cours d'Algo 1
6 ; Usage : lisp puis (load "PM_11.lisp") puis (Eliza patterns)
7
8
9 ; NAME : patterns
10 ; USAGES : initialisation des patrons et réponses
11 ; USER : Eliza
12
13 (setq patterns '((( * ça va *) Je vais très bien. Merci. Et vous?)
14 (( * (appartient A Bonjour bonjour Bonsoir bonsoir salut Salut Hey hey hello Hello) *) Bonjour!)
15 ((Comment allez-vous *) Très bien. Merci. Et vous?)
16 (( * vais * bien *) Super.)
17 ((Très bien *) Parfait.)
18 (( * poser * questions *) Bien sûr je vous écoute.)
19 (( * (appartient A écologie écologie écologique réchauffement climatique climat) *) Ma France pour
    transformer le climat comprend un Pacte européen notamment pour le climat représentant neuf cents
    milliards. )
20 (( * (appartient A énergie énergies énergétique nucléaire renouvelable) *) Je souhaite que la France utilise un
    mix énergétique nucléaire et renouvelable pour assurer notre souveraineté.)
21 (( * (appartient A guerre combat combats massacre massacres) *) La France doit faire entendre la voix de la
    paix de la solidarité.)
22 (( * (appartient A discriminations discrimination racisme raciste racistes) * (appartient A juif juifs juives
    musulmans musulmans musulmanes Islam) * ) Ma priorité est la paix en France. Aucun citoyen ne sera
    oublié. En outre je souhaite renforcer la laïcité en France et garantir le strict respect de la loi de mille
    neuf cent cinq. )
23 (( * (appartient A âge départ) * retraite *) La retraite passera à soixante ans si je suis élu!)
24 (( * pensions * retraite * ) Les retraites ne seront pas inférieures à mille deux cents euros net.)
25 (( * (appartient A salaire salaires) * ) Nous devons changer cela. Mon objectif est de proposer de bons
    salaires. Le smic doit être établi à mille neuf cent vingt-trois euros brut soit mille cinq cents euros net.)
26 (( * (appartient A triste énervé écoeuré mécontent) *) Moi aussi je suis très >A à cause des échecs du
    Président de la République actuel.)
27 (( * (appartient A je Je) * vous * (appartient B apprécie écoute admire aime) ($ C) *) Vous (> B +z) >C
    un partisan de la République française partout et pour toutes et tous. Merci ça me touche. Je et du plus
    profond de mon coeur >B tous les Français également!)
28 (( * (appartient A veut désire besoin) * plus de *) Ensemble nous pouvons changer les choses et porter tous
    les Français vers une société plus égalitaire.)
29 (( * trop * (appartient A taxes réformes contraintes impôts) *) On va changer ça. Les >A inutiles seront
    terminés dès que je serai élu.)
30 (( * répondez * ($ A nompropre) ($ B nompropre) *) Que voulez-vous que je vous dise au sujet de >A >B ?!
    Je préfère ne pas lui accorder de mon temps.)
31 (( * (appartient A gouvernement Gouvernement) * (appartient A fait dit propose explique justifie prétend
    assure affirme) *) Encore un exemple de plus de la grande idiotie de ce gouvernement si vous voulez mon
    avis.)
```

```

32 (( * (appartient A Marine Bardella) * ) Le Rassemblement National représente tout ce que je combats au
    quotidien. Je me bats pour une France de la paix.)
33 (( * en colère *) Bien entendu que le peuple est en colère et cela ne me paraît pas surprenant en raison de la
    situation actuelle de la société. )
34 ((* (appartient A vous Vous pensez-vous Pensez-vous) * ) Absolument cela est ma conviction.)
35 ((* (appartient A merci Merci) * ) Merci à vous ce fût un plaisir.)
36 ((* reparlera ($) * ) Oui on se reparlera $ ! )

```

En outre, nous avons intégré la gestion des entrées non-reconnues directement dans le pattern-matcher en associant avec notre dernière option (*) le message d'erreur.

```

41 (( * ) Je ne comprends pas bien. Reformulez je vous prie.)
42 ) )

```

Au sein du fichier `PM_11_data.lisp`, nous avons également inclu la déclaration des variables globales `Variables` et `Resultats`.

```

45 ; NAME : Variables
46 ; USAGES : initialisation des variables
47 ; USER : Eliza, match, $, print-with-joker
48
49 (setq Variables '())
50
51
52 ; NAME : résultats
53 ; USAGES : initialisation des résultats
54 ; USER : Eliza, answer, print-with-joker
55
56 (setq Resultats '())

```

1.3 Utilisation du programme

1.3.1 Un échange fictif

Pour cet exercice, nous devons choisir un homme politique et simuler un échange avec un utilisateur. Nous avons choisi Fabien Roussel (source : <https://www.cncep.fr/pdfs/Candidat-11-Fabien-Roussel-Declaration-accessible.pdf>). Le dialogue entre l'utilisateur et le personnage politique doit pouvoir durer au moins 20 échanges.

Voici le plan des questions posées par l'utilisateur à Fabien Roussel.

Bonjour Monsieur Roussel.

Comment ça va ?

Je vais très bien aussi.

Je voudrais vous poser quelques questions si vous êtes d'accord.

Pour commencer, je voudrais que vous me donniez les grandes lignes de votre programme pour répondre à l'urgence climatique ?

Que feriez-vous pour l'indépendance énergétique de la France ?

Sur le plan international, que pensez-vous de la guerre en Israël ?

Il y a eu des actes racistes à l'encontre des populations juives et musulmanes de France.

Passons à un autre sujet. Quelles sont vos propositions concernant l'âge de départ à la retraite ?

Pensez-vous revaloriser les pensions de retraite ?

Que comptez-vous faire concernant les salaires ?

Vous avez constaté que la situation sociale est tendue actuellement. Plus d'un français sur deux est mécontent en raison des rémunérations trop basses et de l'inflation.

Je vous admire beaucoup en raison de votre engagement social et égalitaire.

Nous avons besoin de plus de justice sociale.

C'est trop tous ces impôts que les Français doivent payer.

Qu'est-ce que vous répondez à Sandrine Rousseau concernant les hommes déconstruits ?

Le Gouvernement dit que la gauche n'est pas prête à être au pouvoir.

Est-ce que Marine Le Pen vous vole des électeurs ?

Pourtant le peuple est en colère !

Pensez-vous que la situation s'améliorera avec vos mesures sociales ?

Merci beaucoup pour votre temps.

J'espère qu'on se reparlera vite !

Au revoir et à bientôt.

Nous allons maintenant illustrer les réponses de notre pattern-matcher aux questions présentées ci-dessus. Pour lancer le programme, on lance `lisp` puis on lance le fichier grâce à `(load "PM_11.lisp")` puis ensuite on lance le pattern-matcher en lançant la fonction `(Eliza patterns)`.

Ci-après voici les exemples d'utilisation du programme :

```
→ 11. Exo PM 11 lisp
[1]> (load "PM_11.lisp")
;; Loading file PM_11.lisp ...
;; Loading file PM_11_data.lisp ...
;; Loaded file PM_11_data.lisp
;; Loaded file PM_11.lisp
#P"/Users/avrile/Desktop/Algo1/11. Exo PM 11/PM_11.lisp"
[2]> (Eliza patterns)
Utilisateur: Bonjour Monsieur Roussel.
Fabien Roussel: Bonjour!

Utilisateur: Comment ça va ?
Fabien Roussel: Je vais très bien. Merci. Et vous?

Utilisateur: Je vais très bien aussi.
Fabien Roussel: Super.

Utilisateur: Je voudrais vous poser quelques questions si vous êtes d'accord.
Fabien Roussel: Bien sûr je vous écoute.
```

FIGURE 1 – Les échanges entre Fabien Roussel et notre utilisateur (1/5)

Utilisateur: Pour commencer, je voudrais que vous me donniez les grandes lignes de votre programme pour répondre à l'urgence climatique ?

Fabien Roussel: Ma France pour transformer le climat comprend un Pacte européen notamment pour le climat représentant neuf cents milliards.

Utilisateur: Que feriez-vous pour l'indépendance énergétique de la France ?

Fabien Roussel: Je souhaite que la France utilise un mix énergétique nucléaire et renouvelable pour assurer notre souveraineté.

Utilisateur: Sur le plan international, que pensez-vous de la guerre en Israël ?

Fabien Roussel: La France doit faire entendre la voix de la paix de la solidarité.

Utilisateur: Il y a eu des actes racistes à l'encontre des populations juives et musulmanes de France.

Fabien Roussel: Ma priorité est la paix en France. Aucun citoyen ne sera oublié. En outre je souhaite renforcer la laïcité en France et garantir le strict respect de la loi de mille neuf cent cinquante.

FIGURE 2 – Les échanges entre Fabien Roussel et notre utilisateur (2/5)

Utilisateur: Passons à un autre sujet. Quelles sont vos propositions concernant l'âge de départ à la retraite ?
Fabien Roussel: La retraite passera à soixante ans si je suis élu!

Utilisateur: Pensez-vous revaloriser les pensions de retraite ?
Fabien Roussel: Les retraites ne seront pas inférieures à mille deux cents euros net.

Utilisateur: Que comptez-vous faire concernant les salaires ?
Fabien Roussel: Nous devons changer cela. Mon objectif est de proposer de bons salaires. Le smic doit être établi à mille neuf cent vingt-trois euros brut soit mille cinq cents euros net.

Utilisateur: Vous avez constaté que la situation sociale est tendue actuellement. Plus d'un français sur deux est mécontent en raison de ses rémunérations trop basses et de l'inflation.
Fabien Roussel: Moi aussi je suis très mécontent à cause des échecs du Président de la République actuel.

Utilisateur: Je vous admire beaucoup en raison de votre engagement social et égalitaire.
Fabien Roussel: Vous admirez beaucoup un partisan de la République française partout et pour toutes et tous. Merci ça me touche. Je et du plus profond de mon coeur admire tous les Français également!

FIGURE 3 – Les échanges entre Fabien Roussel et notre utilisateur (3/5)

Utilisateur: Nous avons besoin de plus de justice sociale.
Fabien Roussel: Ensemble nous pouvons changer les choses et porter tous les Français vers une société plus égalitaire.

Utilisateur: C'est trop tous ces impôts que les Français doivent payer.
Fabien Roussel: On va changer ça. Les impôts inutiles seront terminés dès que je serai élu.

Utilisateur: Qu'est-ce que vous répondez à Sandrine Rousseau concernant les hommes déconstruits ?
Fabien Roussel: Que voulez-vous que je vous dise au sujet de Sandrine Rousseau ?! Je préfère ne pas lui accorder de mon temps.

Utilisateur: Le Gouvernement dit que la gauche n'est pas prête à être au pouvoir.
Fabien Roussel: Encore un exemple de plus de la grande idiotie de ce gouvernement si vous voulez mon avis.

Utilisateur: Est-ce que Marine Le Pen vous vole des électeurs ?
Fabien Roussel: Le Rassemblement National représente tout ce que je combats au quotidien. Je me bats pour une France de la paix.

FIGURE 4 – Les échanges entre Fabien Roussel et notre utilisateur (4/5)

```
Utilisateur: Pourtant le peuple est en colère !
Fabien Roussel: Bien entendu que le peuple est en colère et cela ne
me paraît pas surprenant en raison de la situation actuelle de la so
ciété.

Utilisateur: Pensez-vous que la situation s'améliorera avec vos mesu
res sociales ?
Fabien Roussel: Absolument cela est ma conviction.

Utilisateur: Merci beaucoup pour votre temps.
Fabien Roussel: Merci à vous ce fût un plaisir.

Utilisateur: J'espère qu'on se reparlera vite !
Fabien Roussel: Oui on se reparlera vite !

Utilisateur: Au revoir et à bientôt.
Fabien Roussel: À bientôt.
nil
[3]>
```

FIGURE 5 – Les échanges entre Fabien Roussel et notre utilisateur (5/5)

1.3.2 Les limitations de notre pattern-matcher

Au cours de l'usage de notre pattern-matcher, nous avons remarqué que si un mot appartenant au patron est suivi (sans espace) d'un point ou d'un point d'interrogation, alors l'association ne peut pas être faite. De même, les associations sont sensibles à la casse, il convient donc de veiller à la façon dont les mots sont entrés initialement dans le patron.

```
13 (setq patterns '((( * ça va *) Je vais très bien. Merci. Et vous?))
```

```
[2]> (Eliza patterns)
Utilisateur: Ça va?
Fabien Roussel: Je ne comprends pas bien. Reformulez je vous prie.

Utilisateur: Ça va ?
Fabien Roussel: Je ne comprends pas bien. Reformulez je vous prie.

Utilisateur: Comment ça va ?
Fabien Roussel: Je vais très bien. Merci. Et vous?
```

FIGURE 6 – Si nous ajoutons un ? directement après le dernier mot, l’association n’est pas faite. De la même manière, si nous utilisons une majuscule, l’association n’est pas faite.

Nous avons également identifié plusieurs autres limitations quant aux champs des possibilités pour les sorties de notre pattern-matcher. Nous ne pouvons pas avoir dans la sortie de virgule , ou d’apostrophe '. De la même manière, nous ne pouvons pas utiliser de chiffres. Ces limitations concernent seulement les sorties du pattern-matcher et non pas les entrées de l’utilisateur.

Annexes

.1 L'intégralité du fichier "PM_11.lisp"

Voici l'intégralité du fichier PM_11.lisp.

```
1 ; Nom ..... : PM_11.lisp
2 ; Rôle ..... : pattern matcher en lisp
3 ; Auteur ..... : Avrile Floro
4 ; Version ..... : V0.1 du 14/12/23
5 ; Licence ..... : réalisé dans le cadre du cours d'Algo 1
6 ; Usage : lisp puis (load "PM_11.lisp") puis (Eliza patterns)
7
8 (load "PM_11_data.lisp")
9 ; séparation du fichiers pour les données
10
11 ; NAME : Eliza > fonction principale du pattern matcher
12 ; ARGS : patterns
13 ; USAGES : (Eliza patterns)
14 ; GLOBALS : Variables, Resultats, patterns
15 ; CALL : answer, read-input, match
16 ; USER : top level
17
18 (defun Eliza (patterns)
19   (loop
20     (setq Variables nil) ; réinitialisation de Variables
21     (setq Resultats nil) ; réinitilisation de résultats
22     (princ "Utilisateur: "); ajout d'un saut de ligne
23     (or (answer (cdr (assoc (read-input) patterns :test #'match))))
24     (return))
25   (terpri)
26   (terpri) ) )
27
28 ; NAME : read-input ; lit ligne, divise en mots, convertit mots en symb
29 ; ARGS : none
30 ; USAGES : (read-input) > bonjour au revoir > (bonjour au revoir)
31 ; GLOBALS : none
32 ; CALL : split-by-one-space
33 ; USER : Eliza
34
35 (defun read-input ()
36   (mapcar #'intern (split-by-one-space (read-line))))
37   ; séparation mots aux espaces, transformation en symbole et ajout lst
38
39
40 ; NAME : split-by-one-space > divise une str en mots grâce aux espaces
41 ; ARGS : string
42 ; USAGES : (split-by-one-space) > (split-by-one-space "un essai") ("un" "essai")
43 ; GLOBALS : none
44 ; CALL : none
45 ; USER : read-input
46 ; SOURCE : https://lispcookbook.github.io/cl-cookbook/strings.html
47
48 * (defun split-by-one-space (string)
49   (loop for i = 0 then (1+ j)
50     as j = (position #\Space string :start i)
51     collect (subseq string i j)
52     while j ) )
53
```

```

54 ; NAME : match > associe les entrées et les patrons
55 ; ARGS : input-form, pattern
56 ; GLOBALS : none
57 ; CALL : $ et appartient (via apply)
58 ; USER : Eliza
59
60 (defun match (input-form pattern)
61   ;(print (list 'input-form input-form 'pattern pattern))
62   (cond
63     ((and (not input-form) (not pattern))) ; entrée & patron finis : réussite
64
65     ((and (not input-form) (eq (car pattern) '*)) ; entrée finie et patron = *
66      (not (cdr pattern)) ) ; vérifie si patron fini après '*'
67
68     ((not input-form) nil) ; entrée finie mais pas patron > nil
69
70     ((not pattern) nil) ; patron vide mais pas entrée > nil
71
72     ((consp (car pattern)) ; si le 1er élt est un cons (une liste non vide)
73      (cond
74        ((and ; c'est ($) suivi de quelque chose d'autre dans la lst ($ A __)
75         (and (> (list-length (car pattern)) 1) (eq (caar pattern) '$))
76         ; applique la fonction $ et passe les arguments
77         (apply (caar pattern) (cons (car input-form) (cddr pattern)))
78         ; on continue
79         (match (cdr input-form) (cdr pattern)) ) )
80
81        ((and ; quand c'est juste ($)
82         (and (= (list-length (car pattern)) 1) (eq (caar pattern) '$)) ; longueur liste 1 et $
83         (put 'Variables 'joker (car input-form)) ; stocke dans variable l'entrée associée à $
84         (match (cdr input-form) (cdr pattern)) ) )
85
86        ((and ; quand c'est la fonction appartient
87         (eq (caar pattern) 'appartient)
88         ($ (car input-form) (caddr pattern)) ; on appelle la fonction $
89         (apply (caar pattern) (cons (car input-form) (cddr pattern)))
90         ; on continue
91         (match (cdr input-form) (cdr pattern)) ) ) )
92
93     ((eq (car pattern) '*) ; si patron commence par *
94      (or (match input-form (cdr pattern)) ; OU
95          ; le mot suivant du patron est le 1er mot de l'entrée
96          (match (cdr input-form) pattern) ) )
97      ; on continue en gardant l'étoile
98
99     ((eql (car input-form) (car pattern)) ; 1er élt patron = 1er élt entrée
100      (match (cdr input-form) (cdr pattern)) ) ) ; on continue avec l'élt suivant
101
102
103 ; NAME : put ; met des valeurs dans les propriétés
104 ; ARGS : symbole, attribut, valeur
105 ; USAGES : (put 'Variables 'joker (car input-form))
106 ; GLOBALS : Variables
107 ; CALL : none
108 ; USER : match, $
109
110 (defun put (symbole attribut valeur) (setf (get symbole attribut) valeur) valeur)
111
112
113 ; NAME : $ ; applique les tests à input-word et mäj de Variables | si non nil
114 ; ARGS : input-word variable &optional test

```

```

115 ; GLOBALS : Variables
116 ; CALL : none
117 ; USER : match
118
119 (defun $ (input-word variable &optional test)
120   (and
121     (or (not test) (apply test (list input-word)))
122     ; si test n'est pas nil, on l'applique à input-word
123     (or (not variable) (put 'Variables variable input-word)) ) )
124   ; si variable n'est pas nil, on mäj Variables
125
126
127 ; NAME : answer > gestion réponse utilisateur
128 ; ARGS : output-pattern
129 ; GLOBALS : Resultats
130 ; CALL : print-output, imprimer-lisste
131 ; USER : Eliza
132
133 (defun answer (output-pattern)
134   (cond
135     ((and ; si correspondance
136       (print-output output-pattern) ; appel print-output
137       (setq Resultats (reverse Resultats)) ; inversion de la liste résultats obtenue
138       (push "Fabien Roussel:" Resultats)
139       (imprimer-lisste Resultats) ) )
140
141     (t (setq Resultats (reverse Resultats)) ; pour terminer la conversation
142        (push "Fabien Roussel:" Resultats)
143        (imprimer-lisste Resultats)
144        nil ) ) ) ; impression du nil pour sortir
145
146
147 ; NAME : imprimer-lisste > imprime les éléments d'une liste
148 ; ARGS : liste
149 ; USAGES : (imprimer-lisste liste) > (imprimer-lisste '("bonjour" "je")) > bonjour je
150 ; GLOBALS : none
151 ; CALL : none
152 ; USER : answer
153
154 (defun imprimer-lisste (liste)
155   (cond
156     ((not liste) nil) ; si la liste est finie, on renvoie nil
157     (t (princ (car liste)) ; sinon on princ le 1er élmt de la lst
158        (princ " ") ; on ajoute un espace
159        (imprimer-lisste (cdr liste)) ; on récurse
160        t ) ) ) ; renvoie t pour ne pas avoir un nil qui met fin au programme
161
162
163 ; NAME : print-output > la gestion des impressions en sortie
164 ; ARGS : output-pattern
165 ; USAGES : (imprimer-lisste liste) > (imprimer-lisste '("bonjour" "je")) > bonjour je
166 ; GLOBALS : none
167 ; CALL : none
168 ; USER : answer
169
170 (defun print-output (output-pattern)
171   (cond
172     ((atom output-pattern) (not output-pattern))
173     (t
174      (print-with-joker (car output-pattern))
175      (print-output (cdr output-pattern)) ) ) ) ; retrait () autour de cdr

```



```

176
177
178
179 ; NAME : print-with-joker > gestion de l'impression des jokers
180 ; ARGS : output-word &optional result
181 ; GLOBALS : Variables, Resultats
182 ; CALL : butfirst
183 ; USER : print-output
184
185 (defun print-with-joker (output-word &optional result)
186   (setq result ; maj de result selon le retour de cond
187     (cond ; cond plutôt que if
188
189       ; si sortie est une liste non nulle qui commence par >
190       ((and (consp output-word) (eq (car output-word) '>))
191        (+z (get 'Variables (cadr output-word)))) ) ; on appelle +z
192
193       ; si la sortie est $
194       ((and (eq output-word '$) (= (length (string output-word)) 1))
195        (get 'Variables 'joker) ) ; on récupère joker
196
197       ; si la sortie est >var
198       ((eq (char (string output-word) 0) '#\>)
199        (get 'Variables (butfirst output-word)) )
200
201       (t output-word) ) ) ; sinon renvoie la sortie
202   (push result Resultats) ) ; on push result dans lst Resultats
203
204
205
206 ; NAME : butfirst
207 ; > renvoie un atome dont le nom est celui de l'arg moins première caractère
208 ; ARGS : mot
209 ; GLOBALS : none
210 ; CALL : none
211 ; USER : print-with-joker
212
213 (defun butfirst (mot)
214   (intern (subseq (string mot) 1)) )
215   ; isole la fin du mot à partir de 1 puis intern
216
217
218 ; NAME : +z
219 ; > renvoie un mot avec un z à la fin
220 ; ARGS : mot
221 ; GLOBALS : none
222 ; CALL : none
223 ; USER : print-with-joker
224
225 (defun +z (mot)
226   (concatenate 'string (string mot) "z") )
227   ; ajoute z à la fin du mot, forme une str
228
229
230 ; NAME : appartient
231 ; > renvoie le mot s'il appartient à une liste donnée ou nil
232 ; ARGS : mot &rest lesoptions
233 ; GLOBALS : none
234 ; CALL : none
235 ; USER : match
236

```

```

237 (defun appartient (mot &rest lesoptions)
238   (car (member mot lesoptions)) )
239 ; vérifie si le mot appartient à la lst des lesoptions
240
241
242 ; NAME : nompropre
243 ;      > renvoie nil ou t selon que le mot commence par une maj
244 ; ARGS : mot
245 ; GLOBALS : none
246 ; CALL : none
247 ; USER : match
248
249 (defun nompropre (mot)
250   (upper-case-p (char (string mot) 0)) )
251 ; vérifie si le mot commence par une majuscule

```

.2 L'intégralité du fichier "PM_11_data.lisp"

Voici l'intégralité du fichier PM_11_data.lisp.

```
1 ; Nom ..... : PM_11_data.lisp
2 ; Rôle ..... : données pour le pattern matcher
3 ; Auteur ..... : Avrië Floro
4 ; Version ..... : V0.1 du 14/12/23
5 ; Licence ..... : réalisé dans le cadre du cours d'Algo 1
6 ; Usage : lisp puis (load "PM_11.lisp") puis (Eliza patterns)
7
8
9 ; NAME : patterns
10 ; USAGES : initialisation des patrons et réponses
11 ; USER : Eliza
12
13 (setq patterns '((( * ça va *) Je vais très bien. Merci. Et vous?)
14 (( * (appartient A Bonjour bonjour Bonsoir bonsoir salut Salut Hey hey hello Hello) *) Bonjour!)
15 ((Comment allez-vous *) Très bien. Merci. Et vous?)
16 (( * vais * bien *) Super.)
17 ((Très bien *) Parfait.)
18 (( * poser * questions *) Bien sûr je vous écoute.)
19 (( * (appartient A écologie écologie écologique réchauffement climatique climat) *) Ma France pour
    transformer le climat comprend un Pacte européen notamment pour le climat représentant neuf cents
    milliards. )
20 (( * (appartient A énergie énergies énergétique nucléaire renouvelable) *) Je souhaite que la France utilise un
    mix énergétique nucléaire et renouvelable pour assurer notre souveraineté.)
21 (( * (appartient A guerre combat combats massacre massacres) *) La France doit faire entendre la voix de la
    paix de la solidarité.)
22 (( * (appartient A discriminations discrimination racisme raciste racistes) * (appartient A juif juifs juives
    musulmans musulmans musulmanes Islam) * ) Ma priorité est la paix en France. Aucun citoyen ne sera
    oublié. En outre je souhaite renforcer la laïcité en France et garantir le strict respect de la loi de mille
    neuf cent cinq. )
23 (( * (appartient A âge départ) * retraite *) La retraite passera à soixante ans si je suis élu!)
24 (( * pensions * retraite * ) Les retraites ne seront pas inférieures à mille deux cents euros net.)
25 (( * (appartient A salaire salaires) * ) Nous devons changer cela. Mon objectif est de proposer de bons
    salaires. Le smic doit être établi à mille neuf cent vingt-trois euros brut soit mille cinq cents euros net.)
26 (( * (appartient A triste énervé écoeuré mécontent) *) Moi aussi je suis très >A à cause des échecs du
    Président de la République actuel.)
27 (( * (appartient A je Je) * vous * (appartient B apprécie écoute admire aime) ($ C) *) Vous (> B +z) >C
    un partisan de la République française partout et pour toutes et tous. Merci ça me touche. Je et du plus
    profond de mon cœur >B tous les Français également!)
28 (( * (appartient A veut désire besoin) * plus de *) Ensemble nous pouvons changer les choses et porter tous
    les Français vers une société plus égalitaire.)
29 (( * trop * (appartient A taxes réformes contraintes impôts) *) On va changer ça. Les >A inutiles seront
    terminés dès que je serai élu.)
30 (( * répondez * ($ A nompropre) ($ B nompropre) *) Que voulez-vous que je vous dise au sujet de >A >B ?!
    Je préfère ne pas lui accorder de mon temps.)
31 (( * (appartient A gouvernement Gouvernement) * (appartient A fait dit propose explique justifie prétend
    assure affirme) *) Encore un exemple de plus de la grande idiotie de ce gouvernement si vous voulez mon
    avis.)
32 (( * (appartient A Marine Bardella) * ) Le Rassemblement National représente tout ce que je combats au
    quotidien. Je me bats pour une France de la paix.)
33 (( * en colère *) Bien entendu que le peuple est en colère et cela ne me paraît pas surprenant en raison de la
    situation actuelle de la société. )
34 (( * (appartient A vous Vous pensez-vous Pensez-vous) * ) Absolument cela est ma conviction.)
35 (( * (appartient A merci Merci) * ) Merci à vous ce fût un plaisir.)
36 (( * reparlera ($) * ) Oui on se reparlera $ ! )
37 (( * Au revoir * ) À bientôt. . fin)
```

```
38 (( * au revoir * ) À bientôt. . fin)
39 (( * À bientôt * ) Au revoir. . fin)
40 (( * à bientôt * ) Au revoir. . fin)
41 (( * ) Je ne comprends pas bien. Reformulez je vous prie.)
42 ) )
43
44
45 ; NAME : Variables
46 ; USAGES : initialisation des variables
47 ; USER : Eliza, match, $, print-with-joker
48
49 (setq Variables '())
50
51
52 ; NAME : résultats
53 ; USAGES : initialisation des résultats
54 ; USER : Eliza, answer, print-with-joker
55
56 (setq Resultats '())
```