

Comment créer un bot ?

Tout chatterbot se compose de deux éléments élémentaires : une base de connaissances et un moteur qui les traite. Cet article présente les bases de AIML, langage de description des connaissances. Il a été conçu dans le cadre du projet de l'Intelligence Artificielle ALICE et il est à la base du fonctionnement de tous les bots qui reposent sur ce projet ouvert. Vous allez créer un bot tout simple qui accueille des utilisateurs et qui les informe sur les offres de la vitrine Internet de Software 2.0. Les bots de ce type deviennent de plus en plus populaires sur Internet. Pour créer le bot, vous allez vous servir du moteur de traitement des connaissances PyAIML, écrit en Python. Néanmoins, la base de connaissances en AIML peut être utilisée dans tout bot qui est conforme au standard de ALICE (liste d'implémentations gratuites de ALICE pour des langues différentes est disponible sur le site <http://www.alicebot.org/downloads>). Le CD inclu contient tout outil nécessaire pour créer et lancer le bot discutant en français. Alors, au boulot, les botmasters !

Interface rapide du bot

PyAIML n'est qu'un moteur de traitement de AIML et il n'est pas doté d'une interface d'utilisateur. Avant de créer une base de connaissances, il faut donc écrire un court script en Python qui récupère les messages de l'utilisateur et affiche les réponses du bot. Le Listing 1 présente l'exemple de ce script. Il importe d'abord la bibliothèque *aiml*, lance ensuite l'interpréteur AIML qu'elle rend disponible et lit le fichier de connaissances sélectionné (ici, fichier *devbot.aiml*). Le bot lui-même ne fait que récupérer les phrases de l'utilisateur dans une boucle infinie et écrire les réponses générées par le module *aiml*. Pour terminer le travail de ce script, il faut appuyer sur [Ctrl-C]. Vous améliorez cet élément plus tard ; pour l'instant, il faut créer le fichier AIML : base de connaissances de votre bot.

Questions - réponses

AIML version 1.0 est conforme au standard XML. Tous ceux qui ont déjà vu XML ou HTML n'auront donc aucun problème pour construire un fichier AIML. Le fichier AIML est un fichier texte qui contient des balises à une structure définie : le prologue standard XML se trouve en tête du script et le contenu du fichier (base de connaissances de votre bot) - entre les balises `<aiml></aiml>`. Les commentaires se situent entre les balises `<!--...-->`.

En AIML, les balises élémentaires de connaissances sont les balises `<category>` à l'intérieur desquelles sont placées les balises `<pattern>` qui con-

Démarrage rapide

Dans cet article, vous allez vous servir d'un moteur de bot, conçu dans le cadre du projet PyAIML et écrit en Python. Pour pouvoir l'utiliser, il faut d'abord installer l'interpréteur de Python (utilisateurs de Linux l'ont probablement déjà chez eux) et le module PyAIML. Le CD inclus contient les fichiers d'installation de Python et PyAIML sous Windows et Linux. Pour l'installer sous Windows, il suffit de lancer *Python-2.3.4.exe* et *PyAIML-0.8.3.win32.exe*. Pour l'installer sous Linux, il faut décompresser l'archive *PyAIML-0.8.3.tar.gz* pour ajouter le module PyAIML au Python et il faut ensuite lancer le script *setup.py* à l'aide de la commande `python setup.py`. À présent, tous les programmes que vous lancez se servent de la bibliothèque *aiml* qui implémente le langage AIML pour Python. Vous enregistrez les programmes en Python dans les fichiers à l'extension *.py* et vous les lancez à l'aide soit d'un double-clic soit d'une commande `python nom_fichier` à partir de la ligne de commandes (à condition que le répertoire principal de Python contenant le fichier *python.exe* se trouve dans le chemin de système).

Le code complet du bot de l'article se trouve dans le fichier *devbotFR.py* sur le CD et sa base de connaissances – dans le fichier *devbotFR.aiml*. Le répertoire *standard* contient des bases de connaissances AIML standard pour l'anglais. Le script qui s'en sert se trouve dans le fichier *standard.py*.

tiennent les modèles de réponses d'utilisateur et les balises `<template>` qui contiennent celles de bot. Les connaissances du bot se composent donc des éléments structurés ainsi :

```
<category>
  <pattern>
    <!-- modèle de message de l'utilisateur -->
  </pattern>
  <template>
    <!-- réaction du bot au modèle -->
  </template>
</category>
```

Le fonctionnement de AIML se repose sur le modèle stimulus – réponse (en anglais *stimulus-response model*). Comment faire pour que votre bot réagisse à un message d'accueil ? Il faut inscrire *SALUT* entre les balises `<pattern>` et *Bonjour* entre `<template>`. Mais attention : en AIML, toutes les lettres doivent être majuscules et Python n'accepte que les signes de base de l'alphabet latin. Il faut éviter tout signe diacritique.

Le Listing 2 présente le contenu du fichier AIML ainsi créé.

Si vous lancez un bot doté d'une telle base de connaissances, il répondra *Bonjour* lorsque l'utilisateur dira *Salut*. En revanche, il affichera un message d'erreur standard PyAIML lorsqu'il verra tout autre message. Puisque le bot réagit à une phrase correspondant parfaitement au modèle (il ignore les majuscules et les minuscules), il génère une erreur si l'utilisateur écrit p. ex. *Salut, ça va ?* Pour résoudre ce problème, AIML permet de se servir des symboles polysémiques * et _ qui élargissent le fonctionnement des modèles. Pour que votre bot réagisse bien à tout message commençant par le mot *Salut*, il faut donc modifier le modèle de la façon suivante :

```
<pattern>SALUT *</pattern>
```

Maintenant, si l'utilisateur écrit *Salut, mec* ou *Salut, ça va ?*, le bot lui répondra toujours *Bonjour*. En réalité, les façons de dire « bonjour » sont très nombreuses. Pour rendre votre bot plus « humain », il faut ajouter au répertoire d'accueil une option de choix aléatoire d'un message d'accueil. Pour ce faire, vous pouvez utiliser la balise `<random>` qui affiche aléatoirement une de plusieurs réponses de son répertoire. Chaque réponse se trouve entre les balises ``. Vous pouvez étendre les réponses dans la balise à trois messages d'accueil possibles :

```
<template>
  <random>
    <li>Bonjour.</li>
    <li>Salut !</li>
    <li>Bienvenue !</li>
  </random>
</template>
```

Qu'est-ce que votre bot peut faire exactement à l'heure actuelle ? Il réagit à tout message qui commence par le mot *Salut* (et non à un seul mot *Salut*) et il affiche une des réponses possibles. Le Listing 3 présente le code d'une telle base de connaissances. Mais les problèmes d'accueil ne se terminent pas ici. L'utilisateur n'est pas du tout obligé de vous dire *Salut*, il faut donc préparer votre bot à d'autres éventualités.

Listing 1. Script le plus simple d'un bot

```
import aiml
# lancement de l'interpréteur AIML
k = aiml.Kernel()
# lecture du fichier des modèles
k.learn("devbot.aiml")
# boucle principale du programme
while True:
    # récupération des messages de l'utilisateur
    user_input = raw_input("> ")
    # génération des réponses du bot
    answer = k.respond(user_input)
    # affichage des réponses sur l'écran
    print answer
```

Listing 2. Simple modèle de réaction à un message d'accueil

```
<?xml version="1.0" encoding="iso-8859-1"?>
<aiml version="1.0">
<category>
  <pattern>SALUT</pattern>
  <template>Bonjour.</template>
</category>
</aiml>
```

Plusieurs modèles à partir d'un seul

Une des caractéristiques utiles de AIML consiste à pouvoir attribuer plusieurs modèles à un modèle principal. Elle permet de tenir rapidement compte de différentes variantes de réponses de l'utilisateur et de les attribuer à une seule signification de base. Voilà la méthode dont vous avez besoin pour que votre bot réponde de la même manière à des différents messages de l'utilisateur. Vous utilisez donc la balise `<srai>` placée entre les balises `<template>`. La balise `<srai>` contiendra le modèle à qui il faut se référer.

Vous avez donc le modèle élémentaire de réaction à ce message : `SALUT` (Listing 2) et vous avez aussi le modèle élargi : `SALUT *` (Listing 3). Maintenant, il faut les fusionner en remplaçant les réponses du modèle élargi par un lien au modèle élémentaire dans la balise `<srai>`. Il faut aussi ajouter la réaction du bot au mot *Bienvenue* et à toutes les phrases qui commencent par ce mot. Le Listing 4 présente le code de tels modèles. À présent, le bot interprétera de la même manière toute phrase commençant par *Salut* ou *Bienvenue* comme si c'était le mot *Salut* tout seul et il choisira aléatoirement une réponse. De plus, le modèle élémentaire ne doit pas correspondre forcément à une phrase réelle de l'utilisateur. Il peut avoir un nom significatif pour le botmaster (p. ex. `ACCUEIL`) et ne peut servir qu'à un point de repère des autres modèles à l'aide de la balise `<srai>`. C'est une caractéristique fort utile car elle facilite la création de bases de connaissances et rend le code plus clair. Les liens à l'aide des balises `<srai>` peuvent avoir une structure complexe, ce qui permet de créer des bases de connaissances aussi complexes.

Il est temps de voir la partie la plus difficile du travail de tout botmaster, à savoir comment prévoir ce que va dire l'utilisateur. Tout bot doit réagir d'une façon sensée à toute phrase de l'utilisateur qui a du sens. C'est un vrai défi pour les botmasters. Mais il y a plusieurs façons de se débrouiller lorsque le bot ne sait pas quoi répondre. Vous les verrez dans un instant. La base de connaissances dans le fichier `devbotFR.aiml` est enrichie d'autres messages d'accueil mais elle n'est pas exhaustive. Chaque bot devrait avoir le plus grand nombre possible de messages et de réponses pour répondre à tout message possible de l'utilisateur. Les messages d'accueil ne constituent pas une seule partie du bot car il doit, avant tout, savoir répondre aux questions selon son objectif. DevBot sur le CD inclu sait répondre à plusieurs questions concernant Software 2.0 et sa vitrine Internet. Je vous encourage de jeter un coup d'œil sur le fichier `devbotFR.aiml`.

Listing 3. Modèle d'accueil élargi

```
<?xml version="1.0" encoding="iso-8859-1"?>
<aiml version="1.0">

<category>
  <pattern>SALUT */</pattern>
  <template>
    <random>
      <li>Bonjour.</li>
      <li>Salut !</li>
      <li>Bienvenue !</li>
    </random>
  </template>
</category>

</aiml>
```

Variables : mémoire du bot

AIML met à votre disposition des variables où vous pouvez stocker les informations sur l'utilisateur. Pour vous servir de ces variables, vous avez deux balises : `<set>` et `<get>`. Voici la syntaxe de `<set>` :

```
<set name = "nom_variable">valeur de la variable</set>
```

Pour récupérer la valeur d'une variable déclarée, il faut écrire `<get name = "nom_variable"/>`. Les deux balises permettent d'écrire les valeurs de variables sur l'écran. Si vous ne voulez pas afficher les valeurs retournées par ces balises (ou par d'autres), il faut les placer entre les balises `<think></think>`. Tout ce qui s'y trouve est une « réflexion » du bot et n'est pas affiché. Si, par exemple, vous voulez récupérer le prénom de l'utilisateur de son message et y répondre en le répétant (vous verrez dans un instant comment le faire), vous pouvez vous servir de ce modèle de réponse :

```
<template>
  <think><set name="name"><star/></set></think>

  Bienvenue <get name="name"/>!
</template>
```

La balise `<star/>` que vous placez à l'intérieur de `<template>` correspond au symbole `*` utilisé dans les modèles. Si le message ne contient que le prénom Hal, le code ci-dessus l'attribuera à la valeur de la variable `name` et récupérera ensuite cette valeur pour pouvoir afficher la réponse *Bienvenue Hal !* Il est possible d'enregistrer le même code sans les balises `<think>` et `<get>` ici inutiles et récupérer le prénom pour l'afficher sur l'écran.

```
<template>
  Bienvenue <set name="name"><star/></set>!
</template>
```

Discussion

Pour l'instant, votre bot fonctionne selon le principe stimulus-réponse : il voit une phrase et réagit en affichant une réponse. Chaque échange de phrases est traité séparément et il est trop tôt pour parler d'une vraie discussion car cela demande de se rappeler de ce qui a été dit précédemment. Heureusement, AIML met à votre disposition les balises `<that>` et `<topic>` grâce auxquelles le bot peut réagir en fonction de sa dernière réponse et du sujet actuel de la discussion.

La balise `<that>` placée à l'intérieur des catégories est une instruction conditionnelle. Elle permet de répondre selon le modèle de `<template>` seulement si le message précédent du bot correspondait au modèle donné dans `<that>` et le message de l'utilisateur correspondait à celui dans `<pattern>`. La Figure 1 fournit une illustration du principe de fonctionnement de ce mécanisme. La balise `<that>` vérifie la dernière réponse du bot (mots utilisés) et non le modèle selon lequel elle a été donnée.

À présent, vous ajoutez l'option de récupération du prénom de l'utilisateur pour le placer dans un message d'accueil. Vous avez déjà vu la balise `<set>` qui sert à retenir et à afficher le prénom ; maintenant, il faut créer un contexte exact. Le bot demanderait le prénom de l'utilisateur p. ex. lorsque celui-ci lui de-

Listing 4. Modèle élémentaire d'accueil et lien vers ce modèle

```
<?xml version="1.0" encoding="iso-8859-1"?>
<aiml version="1.0">

<category>
  <pattern>SALUT </pattern>
  <template>
    <random>
      <li>Bonjour.</li>
      <li>Salut !</li>
      <li>Bienvenue !</li>
    </random>
  </template>
</category>

<category>
  <pattern>SALUT */</pattern>
  <template><srai>SALUT </srai></template>
</category>

<category>
  <pattern>BIENVENUE</pattern>
  <template><srai>SALUT </srai></template>
</category>

<category>
  <pattern>BIENVENUE */</pattern>
  <template><srai>SALUT</srai></template>
</category>

</aiml>
```

manderait le sien. Il faut s'occuper de deux structures possibles d'une telle question : *Comment t'appelles-tu ? Quel est ton prénom ?* Tout d'abord, il faut créer des modèles correspondants :

```
<category>
  <pattern>QUEL EST TON PRENOM</pattern>
  <template>Je m'appelle DevBot. Et quel est ton prénom ?</
    template>
</category>
<category>
  <pattern>* TU T'APPELLES</pattern>
  <template><srai>QUEL EST TON PRENOM</srai></template>
</category>
```

Comment utiliser le contexte ? Puisque le bot répond de la même manière à deux questions concernant son prénom, il est possible de vérifier cette réponse. Vous ajoutez la balise `<that>` qui vérifie si la dernière réponse correspondait au modèle `* TON PRENOM` (autrement dit, si c'était une réponse qui se terminait par *ton prénom*) :

```
<category>
  <pattern>*</pattern>
  <that>* TON PRENOM</that>
  <template>Bienvenue <set name="name"><star/></set>!</
    template>
</category>
```

Comment fonctionne cette partie du code ? Pour chaque phrase de l'utilisateur (`<pattern>` contient un astérisque donc cela peut être n'importe quoi), le bot vérifie si sa réponse se terminait par *ton prénom*. Si c'est le cas, il retient le prénom de l'utilisateur et affiche le message.

Comment se souvenir du sujet ?

Tous les botmasters doivent accepter le fait qu'ils ne seront jamais capables de prévoir toutes les questions de l'utilisateur et tous leurs contextes. Le langage naturel offre, tout simplement, trop de possibilités. Pour vous aider, vous avez à votre disposition : balise de sujet de conversation `<topic>` et variable de système du même nom. Si la variable `topic` est déclarée et malgré cela, le bot n'est pas capable de répondre au message de l'utilisateur, il cherchera le bloc `<topic>` et piochera une réponse. Comment cela se passe-t-il au niveau d'accueil ? Lorsque deux personnes se voient, elles ne se limitent pas à dire seulement « Bonjour » mais elles continuent et parlent p. ex. de la météo ou de la santé. Vous allez ajouter ces options à votre bot : vous allez déclarer le sujet comme `ACCUEIL` et ajouter le bloc `<topic>` contenant ce sujet-là. Si le bot est désorienté par une phrase inconnue de l'utilisateur, il pourra se servir des réponses du bloc `<topic>`. Tout d'abord, il faut donc ajouter au modèle d'accueil `SALUT` la partie qui déclare une bonne valeur de la variable `topic` :

```
<category>
  <pattern>SALUT </pattern>
  <template>
    <think><set name="topic">ACCUEIL </set></think>
    <random>
```

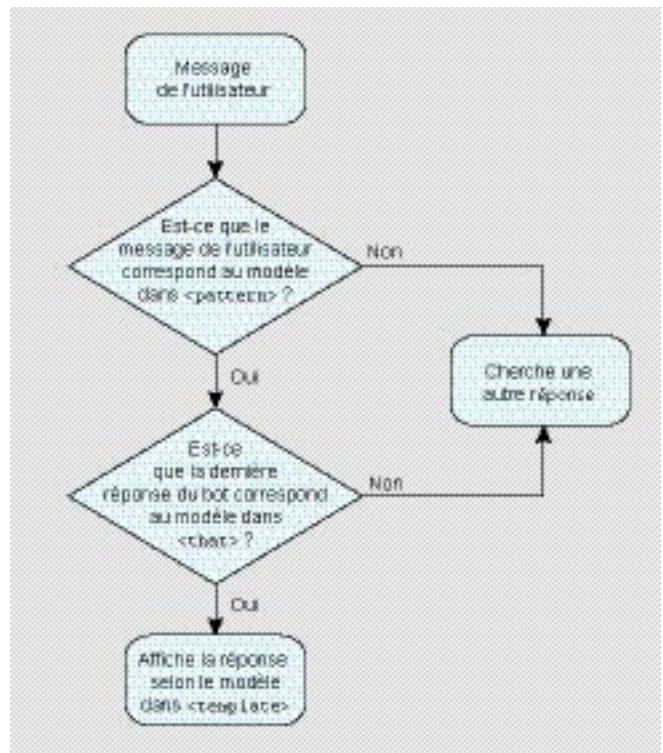


Figure 1. Principe du fonctionnement de la balise `<that>`

```
<li>Bonjour !</li>
<li>Bienvenue !</li>
<li>Salut !</li>
</random>
</template>
</category>
```

Maintenant, vous ajoutez le bloc `<topic>` dans un endroit quelconque du fichier. Ce bloc correspond au sujet de l'accueil (il est recommandé de le placer juste après les modèles d'accueil) :

```
<topic name="ACCUEIL">
  <category>
    <pattern>*</pattern>
    <template>
      <random>
        <li>Quoi de neuf ?</li>
        <li>Ca va?</li>
        <li>En quoi je pourrais t'aider ?</li>
      </random>
    </template>
  </category>
</topic>
```

Vous vous rendez compte que c'est une catégorie toute simple qui a une syntaxe connue. Elle affiche aléatoirement une réponse à chaque question de l'utilisateur et se trouve dans le bloc `<topic>`. Le bot se sert de ce bloc seulement quand il n'est pas capable de répondre à partir des modèles disponibles et quand la valeur de la variable `topic` est déclarée en tant qu'`ACCUEIL`.

Vous avez déjà vu toutes les balises principales et tous les mécanismes essentiels de AIML. Il est temps d'en faire une ba-

Listing 5. Base de connaissances qui se sert de tous les mécanismes présentés dans cet article

```

<?xml version="1.0" encoding="iso-8859-1"?>
<aiml version="1.0">

<category>
  <pattern>SALUT</pattern>
  <template>
    <think><set name="topic">ACCUEIL </set></think>
    <random>
      <li>Bonjour !</li>
      <li>Bienvenue !</li>
      <li>Salut !</li>
    </random>
  </template>
</category>

<category>
  <pattern>SALUT *</pattern>
  <template><srai>SALUT </srai></template>
</category>

<category>
  <pattern>BIENVENUE</pattern>
  <template><srai>SALUT </srai></template>
</category>

<category>
  <pattern>BIENVENUE *</pattern>
  <template><srai>SALUT </srai></template>
</category>

<category>
  <pattern>QUEL EST TON PRENOM</pattern>
  <template>Je m'appelle DevBot. Et quel est ton
    prenom ?</template>
</category>

<category>
  <pattern>* T'APPELLES-TU</pattern>
  <template><srai>QUEL EST TON PRENOM </srai></template>
</category>

<category>
  <pattern>*</pattern>
  <that>* TON PRENOM</that>
  <template>Bienvenue <set name="name"><star></set>!</
    template>
</category>

<topic name="ACCUEIL">
  <category>
    <pattern>*</pattern>
    <template>
      <random>
        <li>Quoi de neuf ?</li>
        <li>Ca va?</li>
        <li>En quoi je pourrais t'aider ?</li>
      </random>
    </template>
  </category>
</topic>
</aiml>

```

se de connaissances qui constituera un fondement pour le développement du bot. Vous pouvez la voir sur le Listing 5. Le fichier *devbotFR.aiml* inclut sur le CD est un exemple de cette base. À part les balises déjà connues, vous y verrez la balise conditionnelle `<condition>` qui permet de décider des réactions du bot en fonction du résultat de la condition. Vous verrez son fonctionnement sur l'exemple d'un modèle qui vérifie si l'utilisateur veut obtenir des renseignements sur Software 2.0 ou sur Software Extra dans le contexte donné. Imaginez que le sujet

de la discussion est déclaré comme `SOFTWARE` ou `SOFTWARE EXTRA`. Le bot demande si l'utilisateur a aimé le numéro précédent et l'utilisateur y répond *oui*. Maintenant, le bot pose une autre question qui commence par *Qu'est-ce que tu as aimé le plus dans le numéro précédent* et qui se termine en fonction du sujet de la conversation. Le code d'une telle situation est le suivant :

```

<category>
  <pattern>OUI</pattern>
  <that>* NUMERO PRECEDENT</that>
  <template>Qu'est-ce que tu as aimé le plus dans le numero
    precedent
    <condition name="topic">
      <li value="SOFTWARE">Software 2.0</li>
      <li value="SOFTWARE EXTRA">Software 2.0 Extra</li>
      <li></li>
    </condition>?
  </template>
</category>

```

Dès que le programme voit la balise `<condition name="topic">`, il vérifie si la valeur de la variable `topic` correspond à la valeur donnée en attribut `value` dans les balises ``. Ces balises sont traitées jusqu'à ce que le programme rencontre une valeur correspondante, ce qui affichera le nom du magazine en question. Si jamais aucune valeur ne correspond, la dernière variable,

Bots sur le Web

Le serveur Pandorabots (<http://www.pandorabots.com>) est très intéressant. Il héberge des bots gratuitement et il met à disposition une interface pour créer, entraîner et modifier des bots sur un navigateur Internet. Après votre inscription et le « log » sur le serveur, vous pouvez concevoir votre bot à partir de rien ou de télécharger votre fichier ou les fichiers AIML. Grâce à l'interface disponible, il est facile de définir les caractéristiques générales du bot, telles que sexe, prénom et de l'entraîner à l'aide d'une conversation dirigée. Le serveur est gratuit et bien documenté et il n'est même pas nécessaire de connaître AIML pour créer et entraîner un bot (mais il est toujours possible de faire des modifications à la main). Il est également possible de placer le bot créé sur un site Web ou de le connecter à un canal IRC choisi ou à Instant Messenger. Les possibilités sont nombreuses.

sans attribut `value`, est effectuée. Dans le code ci-dessus, cette balise est vide donc aucun message ne s'affichera.

Dernières touches

Vous avez créé une base de connaissances complexe (pour sa taille) et qui fonctionne bien. Mais le script à l'aide duquel vous l'alimentiez et testiez n'est pas encore un vrai chatterbot. Pour améliorer leurs bots, les botmasters doivent tout savoir sur leurs erreurs et leurs défauts. Pour cette raison, il est indispensable pour chaque bot de tenir un journal des discussions (logs). DevBot de notre rédaction peut tenir trois logs : log de toutes les discussions, log des remarques pour la rédaction et log des questions (le plus important pour les botmasters) auxquelles le bot n'a pas su répondre. Pour inscrire les remarques dans un log correspondant, il faut se servir de la balise `<system>` dans la base de connaissances. Cette balise permet d'effectuer une commande de système opérationnel placée à l'intérieur de ce système. Dans ce script, cette tâche est réalisée par la catégorie suivante :

```
<category>
  <pattern>*/</pattern>
  <that>QUELS SUJETS */</that>
  <template>Merci beaucoup pour tes remarques. Je vais les
                transmettre au redacteur.
</category>
```

Listing 6. Code complet du bot en Python

```
import aiml
# lancement de l'interpréteur AIML
k = aiml.Kernel()
# lecture du fichier des modèles
k.learn("devbotFR.aiml")
# ouverture/création des fichiers de logs
conv_log = file("conversation.log", "a", 1)
unknown_log = file("unknown.log", "a", 1)
conv_log.write("-----\n")

# boucle principale du programme
while True:
    # récupération des messages de l'utilisateur
    user_input = raw_input("> ")
    if user_input == "quit":
        break
    # définition des réponses
    answer = k.respond(user_input)
    # si la réponse est vide (inconnue)
    # nous écrivons une question au log des inconnus
    if answer == "":
        unknown_log.write(user_input+"\n")
    # enregistrement de la question et de la réponse dans le
        log principal
    conv_log.write(user_input+" : "+answer+"\n")
    # affichage des réponses sur l'écran
    print answer

# fermeture des fichiers de logs
conv_log.close()
unknown_log.close()
```

Sur le Web

- Projet ALICE
<http://www.alicebot.org>
- Documentation AIML
<http://www.alicebot.org/documentation/aiml-reference.html>
- Serveur pour les botmasters où il est possible de créer des bots en ligne
<http://www.pandorabots.com>
- Projet PyAIML
<http://pyaiml.sourceforge.net>
- Base des scripts AIML gratuite
<http://www.aiml.info>
- A-Z List of Chatterbots – une énorme liste de bots disponibles sur Internet
<http://www.angelfire.com/trek/amanda/botlist.htm>

```
<system>echo <star/> >> suggestions.log</system>
</template>
</category>
```

La balise `<system>` n'est exécutée que lorsque la réponse de l'utilisateur commence par les mots *Quels sujets*. Dans ce cas, la commande de système `echo`, commune à Windows et à Linux, ajoute la réponse de l'utilisateur au fichier *suggestions.log*. À part le `<system>`, AIML propose aussi une autre option pour communiquer avec le système extérieur : balise `<javascript>`. Elle est utile pour créer les bots sur les serveurs Internet.

Vous réalisez les deux autres logs dans le code même du bot à l'aide des opérations standard sur les fichiers. Vous ajoutez toutes les réponses enregistrées dans le fichier *conversation.log* au fichier de base et toutes questions auxquelles le bot ne savait pas répondre au fichier *unknown.log*. La dernière modification dans le code du bot consiste à ajouter l'option qui terminera la conversation à l'aide de la commande `quit` pour que l'utilisateur ne soit pas obligé de fermer le programme à l'aide de `[Ctrl-C]`. Le Listing 6 présente le code complet du bot.

Et après ?

Les exemples présentés dans cet article ont pour but de vous montrer les énormes possibilités de AIML et de vous encourager à créer vos bots. C'est une activité fascinante. Le bot que vous venez de créer avec nous est loin d'être complet et sa mise en pratique demande encore du travail. Pour mieux connaître tous les secrets de AIML et toutes les astuces de botmasters, vous trouverez une riche documentation sur Internet. L'encadré *Sur le Web* vous présente les sites Web les plus importants sur ce sujet. Les bots peuvent être utiles p. ex. pour répondre aux questions les plus souvent posées par les clients dans les services clientèle. Mais ils peuvent être également créés pour ressembler le plus possible aux humains. Si vous considérez que votre « petit » est vraiment en forme, vous pouvez l'inscrire à un des concours dont parle Anna Meller dans son article. Seul le manque de l'imagination et de la détermination du botmaster peut limiter la création. Je vous encourage donc à créer votre bot à vous – c'est une aventure formidable. ■