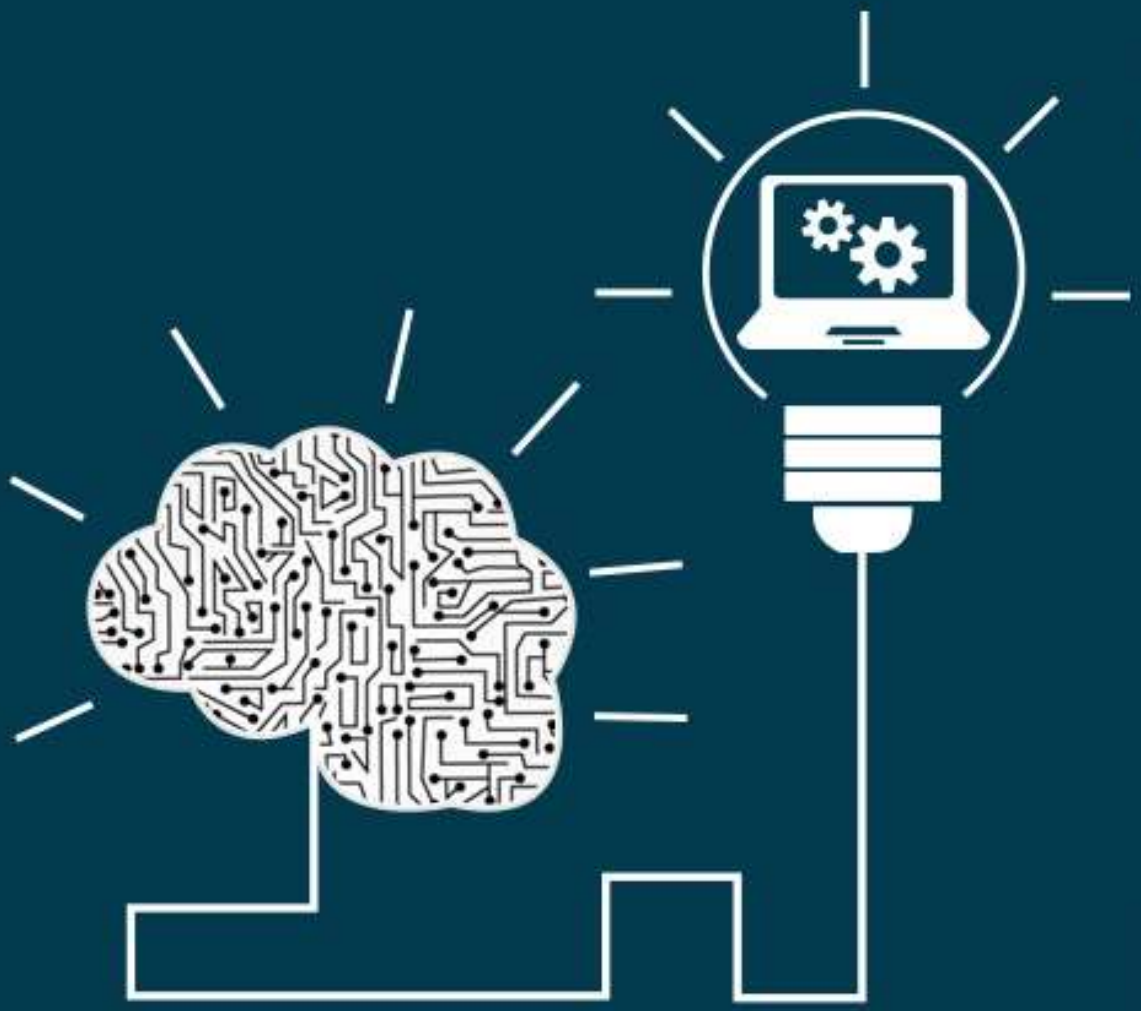

AIML

XML Language

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

AIML stands for Artificial Intelligence Modelling Language. AIML is an XML based markup language meant to create artificial intelligent applications. AIML makes it possible to create human interfaces while keeping the implementation simple to program, easy to understand and highly maintainable.

This tutorial will teach you the basics of AIML. All the basic components of AIML with suitable examples have been discussed in this tutorial.

Audience

This tutorial is designed for software professionals who are willing to learn AIML in simple and easy steps. This tutorial will give you a great understanding on the AIML concepts and after completing this tutorial, you will be at an intermediate level of expertise from where you can take yourself to higher levels of expertise.

Prerequisites

Before proceeding with this tutorial, you should have a basic understanding of Java programming language, because we are going to develop AIML applications using **Program AB**, a Java-based reference implementation of AIML.

Disclaimer & Copyright

Table of Contents

1. AIML – Introduction

AIML stands for **Artificial Intelligence Markup Language**. AIML was developed by the Alicebot free software community and Dr. Richard S. Wallace during 1995-2000. AIML is used to create or customize Alicebot which is a chat-box application based on A.L.I.C.E. (Artificial Linguistic Internet Computer Entity) free software.

AIML Tags

Following are the important tags which are commonly used in AIML documents.

Sr. No.	AIML Tag / Description
1	**<aiml>** Defines the beginning and end of a AIML document.
2	**<category>** Defines the **unit of knowledge** in Alicebot's knowledge base.
3	**<pattern>** Defines the pattern to match what a user may input to an Alicebot.
4	**<template>** Defines the response of an Alicebot to user's input.

We'll discuss each of these tags in AIML Basic tags chapter.

Following are some of the other widely used AIML tags. We'll be discussing each tag in detail in the following chapters.

Sr. No.	AIML Tag / Description
1	**<star>** Used to match wild card * character(s) in the <pattern> Tag.
2	**<srai>** Multipurpose tag, used to call/match the other categories.
3	**<random>** Used **<random>**to get random responses.
4	** ** Used to represent multiple responses.

5	**<set>** Used to set value in an AIML variable.
6	**<get>** Used to get value stored in an AIML variable.
7	**<that>** Used in AIML to respond based on the context.
8	**<topic>** Used in AIML to store a context so that later conversation can be done based on that context.
9	**<think>** Used in AIML to store a variable without notifying the user.
10	**<condition>** Similar to switch statements in programming language. It helps ALICE to respond to matching input.

AIML Vocabulary

AIML vocabulary uses words, space and two special characters * and _ as wild cards. AIML interpreter gives preference to pattern having _ than pattern having *. AIML tags are XML compliant and patterns are case-insensitive.

Example

```
<aiml version="1.0.1" encoding="UTF-8"?>
   <category>
      <pattern> HELLO ALICE </pattern>
      <template>
         Hello User!
      </template>
   </category>
</aiml>
```

Following are the important points to be considered:

- <aiml> tag signifies start of the AIML document.
- <category> tag defines the knowledge unit.
- <pattern> tag defines the pattern user is going to type.
- <template> tag defines the response to the user if user types Hello Alice.

Result

```
User: Hello Alice
Bot: Hello User
```

2. AIML – Environment Setup

This tutorial will guide you on how to prepare a development environment to start your work with AIML to create auto chat software. Program AB is a reference implementation of AIML 2.0 — developed and being maintained by ALICE A.I. Foundation. This tutorial will also teach you how to set up JDK, before you setup Program AB library:

Step 1 – Set up Java Development Kit (JDK)

You can download the latest version of SDK from Oracle's Java site: Java SE Downloads. You will find instructions for installing JDK in downloaded files, follow the given instructions to install and configure the setup. Finally set PATH and JAVA_HOME environment variables to refer to the directory that contains java and javac, typically java_install_dir/bin and java_install_dir respectively.

If you are running Windows and installed the JDK in C:\jdk1.7.0_75, you would have to put the following line in your C:\autoexec.bat file.

```
set PATH=C:\jdk1.7.0_75\bin;%PATH%
set JAVA_HOME=C:\jdk1.7.0_75
```

Alternatively, on Windows NT/2000/XP, you could also right-click on My Computer, select Properties, then Advanced, then Environment Variables. Then, you would update the PATH value and press the OK button.

On Unix (Solaris, Linux, etc.), if the SDK is installed in /usr/local/jdk1.7.0_75 and you use the C shell, you would put the following into your .cshrc file.

```
setenv PATH /usr/local/jdk1.7.0_75/bin:$PATH
setenv JAVA_HOME /usr/local/jdk1.7.0_75
```

Alternatively, if you use an Integrated Development Environment (IDE) like Borland JBuilder, Eclipse, IntelliJ IDEA, or Sun ONE Studio, compile and run a simple program to confirm that the IDE knows where you installed Java, otherwise do proper setup as given document of the IDE.

Step 2 – Set up Program AB

Now if everything is fine, then you can proceed to set up your Program AB. Following are the simple steps to download and install the library on your machine.

- Make a choice whether you want to install AIML on Windows, or Unix and then proceed to the next step to download .zip file.

- Download the latest version of Program AB binaries from https://code.google.com/p/program-ab/ using its program-ab-0.0.4.3.zip link.

- At the time of writing this tutorial, I downloaded **program-ab-0.0.4.3.zip** on my Windows machine and when you unzip the downloaded file it will give you directory structure inside C:\ab as follows.

Sr. No.	Directory	Description
1	c:/ab/bots	Stores AIML bots
2	c:/ab/lib	Stores Java libraries
3	c:/ab/out	Java class file directory
4	c:/ab/run.bat	batch file for running Program AB

Once you are done with this last step, you are ready to proceed with your first AIML Example which you will see in the next chapter.

3. AIML — First Application

Let us start creating first bot which will simply greet a user with **Hello User!** when a user types **Hello Alice**.

Create the Project Structure

As in AIML Environment Setup, we've extracted content of program-ab in **C > ab** with the following directory structure.

Sr. No.	Directory	Description
1	c:/ab/bots	Stores AIML bots
2	c:/ab/lib	Stores Java libraries
3	c:/ab/out	Java class file directory
4	c:/ab/run.bat	batch file for running Program AB

Now, create a directory test inside **C > ab > bots** and create the following directories in it.

Sr. No.	Directory	Description
1	c:/ab/bots/test/aiml	Stores AIML files
2	c:/ab/bots/test/aimlif	Stores AIMLIF files
3	c:/ab/bots/test/config	Stores configuration files
4	c:/ab/bots/test/sets	Stores AIML Sets
5	c:/ab/bots/test/maps	Stores AIML Maps

Create Source Files

Create test.aiml inside **C > ab > bots > test > aiml** and test.aiml.csv inside **C > ab > bots > test > aimlif** directories.

test.aiml

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="1.0.1" encoding="UTF-8"?>
<category>
<pattern> HELLO ALICE </pattern>
<template>
Hello User
</template>
</category>
</aiml>
```

test.aiml.csv

```
0,HELLO ALICE,*,*,Hello User,test.aiml
```

Execute the Program

Open the command prompt. Go to **C > ab >** and type the following command:

```
java -cp lib/Ab.jar Main bot=test action=chat trace=false
```

Verify the Result

You'll see the following output:

```
Working Directory = C:\ab
Program AB 0.0.4.2 beta -- AI Foundation Reference AIML 2.0 implementation
bot=test
action=chat
trace=false
trace mode = false
Name = test Path = C:\ab/bots/test
C:\ab
C:\ab/bots
C:\ab/bots/test
C:\ab/bots/test/aiml
C:\ab/bots/test/aimlif
C:\ab/bots/test/config
C:\ab/bots/test/logs
```

```
C:\ab/bots/test/sets


C:\ab/bots/test/maps


Preprocessor: 0 norms 0 persons 0 person2
Get Properties: C:\ab/bots/test/config/properties.txt
addAIMLSets: C:\ab/bots/test/sets does not exist.
addCategories: C:\ab/bots/test/aiml does not exist.
AIML modified Tue Apr 07 22:24:29 IST 2015 AIMLIF modified Tue Apr 07 22:26:53
IST 2015
No deleted.aiml.csv file found
No deleted.aiml.csv file found
Loading AIML files from C:\ab/bots/test/aimlif
Reading Learnf file
Loaded 1 categories in 0.009 sec
--> Bot test 1 completed 0 deleted 0 unfinished
(1[6])--HELLO-->(1[5])--ALICE-->(1[4])--<THAT>-->(1[3])--*-->(1[2])--<TOPIC>--
>(
1[1])--*-->(0[null,null]) Hello User...
7 nodes 6 singletons 1 leaves 0 shortcuts 0 n-ary 6 branches 0.85714287 average
branching
Human:
```

Type **Hello Alice** and see the result and then type anything else to see the changed result.

```
Human: hello alice
Robot: Hello User
Human: bye
Robot: I have no answer for that.
Human:
```

4. AIML – Basic Tags

In this tutorial, we'll discuss the basic tags of AIML.

- **<aiml>** - defines the beginning and end of a AIML document.
- **<category>** - defines the **unit of knowledge** in Alicebot's knowledge base.
- **<pattern>** - defines the pattern to match what a user may input to an Alicebot.
- **<template>** - defines the response of an Alicebot to user's input.

Following AIML files have been used here as reference.

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="1.0.1" encoding="UTF-8"?>
   <category>
      <pattern> HELLO ALICE </pattern>
      <template>
         Hello User
      </template>
   </category>
</aiml>
```

<aiml> tag

<aiml> tag marks the start and end of a AIML document. It contains version and encoding information under version and encoding attributes. version attribute stores the AIML version used by ALICE chatterbot Knowledge Base, KB. For example, we've used 1.0.1 version. This attribute is optional.

Encoding attributes provide the character sets to be used in the document. For example, we've used UTF-8. As a mandatory requirement, <aiml> tag must contain at least one <category> tag. We can create multiple AIML files where each AIML file contains a single <aiml> tag. The purpose of each AIML file is to add at least a single knowledge unit called category to ALICE chatterbot KB.

```
<aiml version="1.0.1" encoding="UTF-8"?>
...
</aiml>
```

<category> tag

<category> tag is the fundamental knowledge unit of an ALICE Bot. Each category contains:

- User input in the form of a sentence which can be an assertion, question, and exclamation etc. User input can contain wild card characters like *****and **_**.
- Response to user input to be presented by Alicebot.
- Optional context.

A **<category>** tag must have **<pattern>** and **<template>** tag. **<pattern>** represents the user input and template represents the bot's response.

```
<category>
   <pattern> HELLO ALICE </pattern>
   <template>
      Hello User
   </template>
</category>
```

Here, if the user enters **Hello Alice** then bot will respond back as **Hello User.**

<pattern> tag

The <pattern> tag represents a user's input. It should be the first tag within the <category> tag. <pattern> tag can contain wildcard to match more than one sentence as user input. For example, in our example, <pattern> contains HELLO ALICE.

AIML is case-insensitive. If a user enters Hello Alice, hello alice, HELLO ALICE etc., all inputs are valid and bot will match them against HELLO ALICE.

```
<category>
   <pattern> HELLO ALICE </pattern>
   <template>
      Hello User
   </template>
</category>
```

Here, the template is "Hello User" and represents a robot's response to user input.

<template> tag

<template> tag represents the bot's response to the user. It should be the second tag within the <category> tag. This <template> tag can save data, call another program, give conditional answers or delegate to other categories.

```
<category>
   <pattern> HELLO ALICE </pattern>
   <template>
      Hello User
   </template>
</category>
```

Here, the template is "Hello User" and represents a robot's response to the user input.

5. AIML – <star> Tag

<star> Tag is used to match wildcard ***** character(s) in <pattern> Tag.

Syntax

```
<star index = "n"/>
```

n signifies the position of * within the user input in <pattern> Tag.

Consider the following example:

```
<category>
   <pattern> A * is a *. </pattern>
   <template>
      When a <star index="1"/> is not a <star index="2"/>?
   </template>
</category>
```

If the user enters "A mango is a fruit.", then bot will respond as "When a mango is not a fruit?"

Example

Create star.aiml inside **C > ab > bots > test > aiml** and star.aiml.csv inside **C > ab > bots > test > aimlif** directories.

star.aiml

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="1.0.1" encoding="UTF-8"?>
   <category>
      <pattern>I LIKE *</pattern>
      <template>
         I too like <star/>.
      </template>
   </category>
   <category>
      <pattern>A * IS A *</pattern>
   <template>
      How <star index="1"/> can not be a <star index="2"/>?
   </template>
```

```
   </category>
</aiml>
```

star.aiml.csv

```
0,I LIKE *,*,*,I too like <star/>.,star.aiml
0,A * IS A *,*,*,How <star index="1"/> can not be a <star
index="2"/>?,star.aiml
```

Execute the Program

Open the command prompt. Go to **C > ab >** and type the following command:

```
java -cp lib/Ab.jar Main bot=test action=chat trace=false
```

Verify the Result

You will see the following output:

```
Human: I like mango
Robot: I too like mango.
Human: A mango is a fruit
Robot: How mango can not be a fruit?
```

<star index="1"/> is often used as <star />

6. AIML – <srai> Tag

<srai> Tag is a multipurpose tag. This tag enables AIML to define the different targets for the same template.

Syntax

```
<srai> pattern </srai>
```

Following are the commonly used terms associated with **srai**:

- Symbolic Reduction
- Divide and Conquer
- Synonyms resolution
- Keywords detection

Symbolic Reduction

The symbolic reduction technique is used to simplify patterns. It helps to reduce complex grammatical patterns with simple pattern(s).

For example, consider the following conversation.

```
Human: Who was Albert Einstein?
Robot: Albert Einstein was a German physicist.
Human: Who was Isaac Newton?
Robot: Isaac Newton was a English physicist and mathematician.
```

Now **What if** questions are raised as

```
Human: DO YOU KNOW WHO Albert Einstein IS?
Human: DO YOU KNOW WHO Isaac Newton IS?
```

Here, **<srai>** tag works. It can take the pattern of the user as a template.

Step 1: Create categories

```
<category>
   <pattern>WHO IS ALBERT EINSTEIN?</pattern>
   <template>Albert Einstein was a German physicist.</template>
</category>
```

```
<category>
   <pattern> WHO IS Isaac NEWTON? </pattern>
   <template>Isaac Newton was a English physicist and mathematician.</template>
</category>
```

Step 2: Create generic category using <srai> tag

```
<category>
   <pattern>DO YOU KNOW WHO * IS?</pattern>
   <template>
      <srai>WHO IS <star/></srai>
   </template>
</category>
```

Example

Create srai.aiml inside **C > ab > bots > test > aiml** and srai.aiml.csv inside **C > ab > bots > test > aimlif** directories.

srai.aiml

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="1.0.1" encoding="UTF-8"?>
   <category>
      <pattern> WHO IS ALBERT EINSTEIN </pattern>
      <template>Albert Einstein was a German physicist.</template>
   </category>
   <category>
      <pattern> WHO IS Isaac NEWTON </pattern>
      <template>Isaac Newton was a English physicist and
mathematician.</template>
   </category>
   <category>
      <pattern>DO YOU KNOW WHO * IS</pattern>
      <template>
         <srai>WHO IS <star/></srai>
      </template>
   </category>
</aiml>
```

star.aiml.csv

```
0,WHO IS ALBERT EINSTEIN,*,*,Albert Einstein was a German physicist.,srai.aiml
0,WHO IS Isaac NEWTON,*,*,Isaac Newton was a English physicist and
mathematician.,srai.aiml
0,DO YOU KNOW WHO * IS,*,*,<srai>WHO IS <star/></srai>,srai.aiml
```

Execute the Program

Open the command prompt. Go to **C > ab >** and type the following command:

```
java -cp lib/Ab.jar Main bot=test action=chat trace=false
```

Verify the Result

You will see the following output:

```
Human: Do you know who Albert Einstein is
Robot: Albert Einstein was a German physicist.
```

Divide and Conquer

Divide and Conquer is used to reuse sub sentences in making a complete reply. It helps to reduce defining multiple categories.

For example, consider following conversation.

```
Human: Bye
Robot: GoodBye!
Human: Bye Alice!
Robot: GoodBye!
```

Now here robot is expected to reply **GoodBye!** Whenever a user says **Bye** in the beginning of the sentence.

Let's put <srai> tag to work here.

Step 1: Create category

```
<category>
   <pattern>BYE</pattern>
   <template>Good Bye!</template>
</category>
```

Step 2: Create generic category using <srai> tag

```
<category>

   <pattern>BYE *</pattern>

 <template>
      <srai>BYE</srai>
   </template>
</category>
```

Example

Update srai.aiml inside **C > ab > bots > test > aiml** and srai.aiml.csv inside **C > ab > bots > test > aimlif** directories.

srai.aiml

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="1.0.1" encoding="UTF-8"?>
   <category>
      <pattern> WHO IS ALBERT EINSTEIN </pattern>
      <template>Albert Einstein was a German physicist.</template>
   </category>
   <category>
      <pattern> WHO IS Isaac NEWTON </pattern>
      <template>Isaac Newton was a English physicist and
mathematician.</template>
   </category>
   <category>
      <pattern>DO YOU KNOW WHO * IS</pattern>
      <template>
         <srai>WHO IS <star/></srai>
      </template>
   </category>
   <category>
      <pattern>BYE</pattern>
      <template>Good Bye!</template>
   </category>
   <category>
      <pattern>BYE *</pattern>
```

```
      <template>
         <srai>BYE</srai>
      </template>

   </category>
</aiml>
```

star.aiml.csv

```
0,WHO IS ALBERT EINSTEIN,*,*,Albert Einstein was a German physicist.,srai.aiml
0,WHO IS Isaac NEWTON,*,*,Isaac Newton was a English physicist and
mathematician.,srai.aiml
0,DO YOU KNOW WHO * IS,*,*,<srai>WHO IS <star/></srai>,srai.aiml
0,BYE,*,*,Good Bye!,srai.aiml
0,BYE *,*,*,<srai>BYE</srai>,srai.aiml
```

Execute the Program

Open the command prompt. Go to **C > ab >** and type the following command:

```
java -cp lib/Ab.jar Main bot=test action=chat trace=false
```

Verify the Result

You will see the following output:

```
Human: Bye
Robot: GoodBye!
Human: Bye Alice!
Robot: GoodBye!
```

Synonyms Resolution

Synonyms are words with similar meanings. A bot should reply in the same manner for similar words.

For example, consider the following conversation.

```
Human: Factory


Robot: Development Center!
Human: Industry
Robot: Development Center!
```

Now here robot is expected to reply **Development Center!** whenever a user says **Factory** or **Industry**.

Let's put **<srai>** tag to work here.

Step 1: Create category

```
<category>

   <pattern>FACTORY</pattern>
   <template>Development Center!</template>
</category>
```

Step 2: Create generic category using <srai> tag

```
<category>
   <pattern>INDUSTRY</pattern>
   <template>
      <srai>FACTORY</srai>
   </template>
</category>
```

Example

Update srai.aiml inside **C > ab > bots > test > aiml** and srai.aiml.csv inside **C > ab > bots > test > aimlif** directories.

srai.aiml

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="1.0.1" encoding="UTF-8"?>
   <category>
      <pattern> WHO IS ALBERT EINSTEIN </pattern>
      <template>Albert Einstein was a German physicist.</template>
   </category>
   <category>
      <pattern> WHO IS Isaac NEWTON </pattern>
      <template>Isaac Newton was a English physicist and
mathematician.</template>
   </category>
   <category>
      <pattern>DO YOU KNOW WHO * IS</pattern>
      <template>
```

```
         <srai>WHO IS <star/></srai>
      </template>
   </category>
   <category>
      <pattern>BYE</pattern>
      <template>Good Bye!</template>
   </category>
   <category>
      <pattern>BYE *</pattern>
      <template>
         <srai>BYE</srai>
      </template>
   </category>
   <category>
      <pattern>FACTORY</pattern>
      <template>Development Center!</template>
   </category>
   <category>
      <pattern>INDUSTRY</pattern>
      <template>
         <srai>FACTORY</srai>
      </template>
   </category>
</aiml>
```

star.aiml.csv

```
0,WHO IS ALBERT EINSTEIN,*,*,Albert Einstein was a German physicist.,srai.aiml
0,WHO IS Isaac NEWTON,*,*,Isaac Newton was a English physicist and
mathematician.,srai.aiml
0,DO YOU KNOW WHO * IS,*,*,<srai>WHO IS <star/></srai>,srai.aiml
0,BYE,*,*,Good Bye!,srai.aiml
0,BYE *,*,*,<srai>BYE</srai>,srai.aiml
0,FACTORY,*,*,Development Center!,srai.aiml
0,INDUSTRY,*,*,<srai>FACTORY</srai>,srai.aiml
```

Execute the Program

Open the command prompt. Go to **C > ab >** and type the following command:

```
java -cp lib/Ab.jar Main bot=test action=chat trace=false
```

Verify the Result

You will see the following output:

```
Human: Factory
Robot: Development Center!
Human: Industry
Robot: Development Center!
```

Keywords Detection

Using **srai**, we can return a simple response when the user types a specific keyword, say, School, no matter where "school" is present in the sentence.

For example, consider the following conversation.

```
Human: I love going to school daily.
Robot: School is an important institution in a child's life.
Human: I like my school.
Robot: School is an important institution in a child's life.
```

Here, the robot is expected to reply a standard message **'School is an important institution in a child's life.'** whenever a user has **school** in the sentence.

Let's put **<srai>** tag to work here. We'll use wildcards here.

Step 1: Create category

```
<category>
   <pattern>SCHOOL</pattern>
   <template>School is an important institution in a child's life.</template>
</category>
```

Step 2: Create generic categories using <srai> tag

```
<category>
   <pattern>_ SCHOOL</pattern>
   <template>
      <srai>SCHOOL</srai>
   </template>
```

```
</category>
<category>
   <pattern>_ SCHOOL</pattern>
   <template>
      <srai>SCHOOL</srai>
   </template>
</category>
<category>
   <pattern>SCHOOL *</pattern>
   <template>
      <srai>SCHOOL</srai>
   </template>
</category>
<category>
   <pattern>_ SCHOOL *</pattern>
   <template>
      <srai>SCHOOL</srai>
   </template>
</category>
```

Example

Update srai.aiml inside **C > ab > bots > test > aiml** and srai.aiml.csv inside **C > ab > bots > test > aimlif** directories.

srai.aiml

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="1.0.1" encoding="UTF-8"?>
   <category>
      <pattern> WHO IS ALBERT EINSTEIN </pattern>
      <template>Albert Einstein was a german physicist.</template>
   </category>
   <category>
      <pattern> WHO IS Isaac NEWTON </pattern>
      <template>Isaac Newton was a english physicist and
mathematician.</template>
   </category>
   <category>
      <pattern>DO YOU KNOW WHO * IS</pattern>
```

```
      <template>
         <srai>WHO IS <star/></srai>
      </template>
   </category>
   <category>
      <pattern>BYE</pattern>
      <template>Good Bye!</template>
   </category>
   <category>
      <pattern>BYE *</pattern>
      <template>
         <srai>BYE</srai>
      </template>
   </category>
   <category>
      <pattern>FACTORY</pattern>
      <template>Development Center!</template>
   </category>
   <category>
      <pattern>INDUSTRY</pattern>
      <template>
         <srai>FACTORY</srai>
      </template>
   </category>
   <category>
      <pattern>SCHOOL</pattern>
      <template>School is an important institution in a child's
life.</template>
   </category>
   <category>
      <pattern>_ SCHOOL</pattern>
      <template>
         <srai>SCHOOL</srai>
      </template>
   </category>
   <category>
      <pattern>_ SCHOOL</pattern>
      <template>
```

```
         <srai>SCHOOL</srai>
      </template>
   </category>
   <category>
      <pattern>SCHOOL *</pattern>
      <template>
         <srai>SCHOOL</srai>
      </template>
   </category>
   <category>
      <pattern>_ SCHOOL *</pattern>
      <template>
         <srai>SCHOOL</srai>
      </template>
   </category>
</aiml>
```

star.aiml.csv

```
0,WHO IS ALBERT EINSTEIN,*,*,Albert Einstein was a german physicist.,srai.aiml
0,WHO IS Isaac NEWTON,*,*,Isaac Newton was a english physicist and
mathematician.,srai.aiml
0,DO YOU KNOW WHO * IS,*,*,<srai>WHO IS <star/></srai>,srai.aiml
0,BYE,*,*,Good Bye!,srai.aiml
0,BYE *,*,*,<srai>BYE</srai>,srai.aiml
0,FACTORY,*,*,Development Center!,srai.aiml
0,INDUSTRY,*,*,<srai>FACTORY</srai>,srai.aiml
0,SCHOOL,*,*,School is an important institution in a child's life.,srai.aiml
0,_ SCHOOL,*,*,<srai>SCHOOL</srai>,srai.aiml
0,SCHOOL *,*,*,<srai>SCHOOL</srai>,srai.aiml
0,_ SCHOOL *,*,*,<srai>SCHOOL</srai>,srai.aiml
```

Execute the Program

Open the command prompt. Go to **C > ab >** and type the following command:

```
java -cp lib/Ab.jar Main bot=test action=chat trace=false
```

Verify the Result

You will see the following output:

```
Human: I love going to school daily.
Robot: School is an important institution in a child's life.
Human: I like my school.
Robot: School is an important institution in a child's life.
```

7. AIML — <random> Tag

<random> Tag is used to get random responses. This tag enables AIML to respond differently for the same input. <random> tag is used along with <li> tags. tags carry different responses that are to be delivered to the user on a random basis.

Syntax

```
<random>
<li> pattern1 </li>
<li> pattern2 </li>
...
<li> patternN </li>
</random>
```

For example, consider the following conversation.

```
Human: Hi
Robot: Hello!
Human: Hi
Robot: Hi! Nice to meet you!
```

Example

Create random.aiml inside **C > ab > bots > test > aiml** and random.aiml.csv inside **C > ab > bots > test > aimlif** directories.

random.aiml

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="1.0.1" encoding="UTF-8"?>
   <category>
      <pattern>HI</pattern>
      <template>
         <random>
            <li> Hello! </li>
            <li> Hi! Nice to meet you! </li>
         </random>
      </template>
</aiml>
```

random.aiml.csv

```
0,HI,*,*, <random><li> Hello! </li><li> Hi! Nice to meet you!
</li></random>,random.aiml
```

Execute the Program

Open the command prompt. Go to **C > ab >** and type the following command:

```
java -cp lib/Ab.jar Main bot=test action=chat trace=false
```

Verify the Result

You will see the following output:

```
Human: Hi
Robot: Hi! Nice to meet you!
Human: Hi
Robot: Hello!
```

Here, the response may vary considering random responses.

8. AIML — <set>, <get> Tags

<set> and <get> tags are used to work with variables in AIML. Variables can be predefined variables or programmer created variables.

Syntax

<set> tag is used to set value in a variable.

```
<set name = "variable-name"> variable-value </set>
```

<get> tag is used to get value from a variable.

```
<get name = "variable-name"></get>
```

For example, consider the following conversation.

```
Human: I am Mahesh
Robot: Hello Mahesh!
Human: Good Night
Robot: Good Night Mahesh! Thanks for the conversation!
```

Example

Create setget.aiml inside **C > ab > bots > test > aiml** and setget.aiml.csv inside **C > ab > bots > test > aimlif** directories.

setget.aiml

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="1.0.1" encoding="UTF-8"?>
   <category>
      <pattern>I am *</pattern>
      <template>
         Hello <set name="username"> <star/>! </set>
      </template>
   </category>
   <category>
      <pattern>Good Night</pattern>
      <template>
         Hi <get name="username"/> Thanks for the conversation!
```

```
      </template>
   </category>
</aiml>
```

setget.aiml.csv

```
0,I am *,*,*, Hello <set name="username"> <star/>! </set>,setget.aiml
0,Good Night,*,*, Hi <get name="username"/> Thanks for the
conversation!,setget.aiml
```

Execute the Program

Open the command prompt. Go to **C > ab >** and type the following command:

```
java -cp lib/Ab.jar Main bot=test action=chat trace=false
```

Verify the Result

You will see the following output:

```
Human: I am Mahesh
Robot: Hello Mahesh!
Human: Good Night
Robot: Good Night Mahesh! Thanks for the conversation!
```

9. AIML — <that> Tag

<that> Tag is used in AIML to respond based on the context.

Syntax

```
<that> template </that>
```

For example, consider the following conversation.

```
Human: Hi Alice! What about movies?
Robot: Do you like comedy movies?
Human: No
Robot: Ok! But I like comedy movies.
```

Example

Create **that.aiml** inside **C > ab > bots > test > aiml** and that.aiml.csv inside **C > ab > bots > test > aimlif** directories.

that.aiml

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="1.0.1" encoding="UTF-8"?>
   <category>
      <pattern>WHAT ABOUT MOVIES</pattern>
      <template>Do you like comedy movies</template>
   </category>
   <category>
      <pattern>YES</pattern>
      <that>Do you like comedy movies</that>
      <template>Nice, I like comedy movies too.</template>
   </category>
   <category>
      <pattern>NO</pattern>
      <that>Do you like comedy movies</that>
      <template>Ok! But I like comedy movies.</template>
   </category>
</aiml>
```

that.aiml.csv

```
0,WHAT ABOUT MOVIES,*,*,Do you like comedy movies,that.aiml
0,YES,Do you like comedy movies,*,Nice! I like comedy movies too.,that.aiml
0,NO,Do you like comedy movies,*,Ok! But I like comedy movies.,that.aiml
```

Execute the Program

Open the command prompt. Go to **C > ab >** and type the following command:

```
java -cp lib/Ab.jar Main bot=test action=chat trace=false
```

Verify the Result

You will see the following output:

```
Human: What about movies?
Robot: Do you like comedy movies?
Human: No
Robot: Ok! But I like comedy movies.
```

10. AIML — <topic> Tag

<topic> Tag is used in AIML to store a context so that later conversation can be done based on that context. Usually, **<topic>** tag is used in **Yes/No type conversation**. It helps AIML to search categories written within the context of the topic.

Syntax

Define a topic using <set> tag

```
<template> <set name="topic"> topic-name </set></template>
```

Define the category using <topic> tag

```
<topic name="topic-name">
   <category>
         ...
   </category>
</topic>
```

For example, consider the following conversation.

```
Human: let discuss movies
Robot: Yes movies
Human: Comedy movies are nice to watch
Robot: Watching good movie refreshes our minds.
Human: I like watching comedy
Robot: I too like watching comedy.
```

Here bot responds taking "movie" as the topic.

Example

Create topic.aiml inside **C > ab > bots > test > aiml** and topic.aiml.csv inside **C > ab > bots > test > aimlif** directories.

topic.aiml

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="1.0.1" encoding="UTF-8"?>
   <category>
      <pattern>LET DISCUSS MOVIES</pattern>
```

```
      <template>Yes <set name="topic">movies</set></template>
   </category>
   <topic name="movies">
      <category>
         <pattern> * </pattern>
         <template>Watching good movie refreshes our minds.</template>
      </category>
      <category>
         <pattern> I LIKE WATCHING COMEDY! </pattern>
         <template>I like comedy movies too.</template>
      </category>
   </topic>
</aiml>
```

that.aiml.csv

```
0,LET DISCUSS MOVIES,*,*,Yes <set name="topic">movies</set>,topic.aiml
0,*,*,movies,Watching good movie refreshes our minds.,topic.aiml
0,I LIKE WATCHING COMEDY!,*,movies,I like comedy movies too.,topic.aiml
```

Execute the Program

Open the command prompt. Go to **C > ab >** and type the following command:

```
java -cp lib/Ab.jar Main bot=test action=chat trace=false
```

Verify the Result

You will see the following output:

```
Human: let discuss movies
Robot: Yes movies
Human: Comedy movies are nice to watch
Robot: Watching good movie refreshes our minds.
Human: I like watching comedy
Robot: I too like watching comedy.
```

11. AIML — <think> Tag

<think> Tag is used in AIML to store a variable without notifying the user.

Syntax

Store a value using <think> tag

```
<think> <set name="variable-name"> variable-value </set></think>
```

For example, consider the following conversation.

```
Human: My name is Mahesh
Robot: Hello!
Human: Byeee
Robot: Hi Mahesh Thanks for the conversation!
```

Example

Create think.aiml inside **C > ab > bots > test > aiml** and think.aiml.csv inside **C > ab > bots > test > aimlif** directories.

think.aiml

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="1.0.1" encoding="UTF-8"?>
   <category>
      <pattern>My name is *</pattern>
      <template>
         Hello!<think><set name="username"> <star/></set></think>
      </template>
   </category>
   <category>
      <pattern>Byeee</pattern>
      <template>
         Hi <get name="username"/> Thanks for the conversation!
      </template>
   </category>
</aiml>
```

think.aiml.csv

```
0,My name is *,*,*, Hello! <think><set name="username">
<star/></set></think>,think.aiml

0,Byeee,*,*, Hi <get name="username"/> Thanks for the conversation!,think.aiml
```

Execute the Program

Open the command prompt. Go to **C > ab >** and type the following command:

```
java -cp lib/Ab.jar Main bot=test action=chat trace=false
```

Verify the Result

You will see the following output:

```
Human: My name is Mahesh

Robot: Hello!

Human: Byeee

Robot: Hi Mahesh Thanks for the conversation!
```

12. AIML — <condition> Tag

<condition> Tag is similar to switch statements in programming language. It helps ALICE to respond to the matching input.

Syntax

```
<condition name="variable-name" value="variable-value"/>
```

For example, consider the following conversation.

```
Human: How are you feeling today
Robot: I am happy!
```

Here we've stored **happy** as the state of ALICE and that is how it responds as "I am happy!".

Example

Create **condition.aiml** inside **C > ab > bots > test > aiml** and condition.aiml.csv inside **C > ab > bots > test > aimlif** directories.

condition.aiml

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="1.0.1" encoding="UTF-8"?>
   <category>
      <pattern> HOW ARE YOU FEELING TODAY </pattern>
      <template>
         <think><set name="state"> happy</set></think>
         <condition name="state" value="happy">
            I am happy!
         </condition>
         <condition name="state" value="sad">
            I am sad!
         </condition>
      </template>
   </category>
</aiml>
```

condition.aiml.csv

```
0,HOW ARE YOU FEELING TODAY,*,*,<think><set name="state">
happy</set></think><condition name="state" value="happy">I am
happy!</condition><condition name="state" value="sad">I am
sad!</condition>,condition.aiml
```

Execute the Program

Open the command prompt. Go to **C > ab >** and type the following command:

```
java -cp lib/Ab.jar Main bot=test action=chat trace=false
```

Verify the Result

You will see the following output:

```
Human: How are you feeling today
Robot: I am happy!
```