### LABORATORY MANUAL



**DEPARTMENT OF**

**COMPUTER SCIENCE AND ENGINEERING**

U19CS605 – Artificial Intelligence Laboratory

BATCH: 2019- 2023

## SONA COLLEGE OF TECHNOLOGY

## (Autonomous)

## Subject Code : U19CS605

## Subject : Artificial Intelligence Lab

## University : Anna University, Chennai

## Regulation : Autonomous-Regulation 2019

|  | Prepared By | Reviewed By | Approved By |
| --- | --- | --- | --- |
| Name | Dr.K.C.Rajheswari Dr.S.Anitha Elavarasi  Dr.S.Kiruthika  Dr.S.Sankar | Dr.S.Sakthivel | Dr. B. Sathiyabhama |
| Signature |  |  |  |
| Designation | Assistant professorAssistant professorAssistant professorAssistant professor | Professor | HOD/CSE |

**Vision and Mission of the Institute**

Sona College of Technology (SCT) is established in the year 1997. It is one of the nationally ranking self-financing Institutions founded by the Visionary Late. M.S. Chockalingam Chettiar

**Institute Vision and Mission**

| **Vision** |
| --- |

To become an institute of great repute, in the fields of Science, Engineering, Technology and Management studies, by offering a full range of programmes of global standard to foster research, and to transform the students into globally competent personalities

| **Mission** |
| --- |

Sona College of Technology is a private engineering institution that offers engineering degree programmes at under graduate level and post graduate level, computer applications and management studies at post graduate level and doctoral programmes in the areas of engineering and science and humanities.   
The college aims to provide a full-fledged education, to produce graduates with competency to excel in the organizations they serve and to cater to the needs of the community as a whole.   
Our mission for next three years will be

* To offer Graduate, Post-graduate, Doctoral and other value-added programmes beneficial for the students
* To establish state-of-the-art facilities and resources required to achieve excellence in teaching-learning, and supplementary processes
* To provide Faculty and Staff with the required qualification and competence and to provide opportunity to upgrade their knowledge and skills
* To motivate the students to pursue higher education, appear for competitive exams, and other value added programmes for their holistic development
* To provide opportunities to the students and bring out their inherent talent
* To establish Centres of excellence in the emerging areas of research
* To have regular interaction with the Industries in the area of R & D, and offer consultancy, training and testing services
* To offer Continuing education, and Non-formal vocational education programmes that are beneficial to the society
* To inculcate entrepreneurial attitude in the students and to provide a platform to start their own startups in the campus.

**Department Vision and Mission**

| **Vision** |
| --- |

To be the front runner in Computer Science and Engineering education and to foster the students into globally competent professionals with expertise in software development and aptitude for research and ethical values

| **Mission** |
| --- |

The department of Computer Science and Engineering is committed to

**M1**: Provide the ambience to become industry ready Professionals, Researchers and Entrepreneurs by offering courses on cutting edge technology and advanced laboratory courses for the students.

**M2:** Provide a conducive environment for faculty to engage in and train students in progressive and convergent research themes by establishing Centres of Excellence.

**M3:** Impart high quality experiential learning to get expertise in modern software tools and to cater to the real time requirements of the industry.

**M4:** Inculcate problem solving and team building skills and promote lifelong learning with a sense of societal and ethical responsibilities.

**M5:** To offer continuing education programmes in the emerging areas for the benefit of stakeholders**.**

**Program Educational Objectives(PEOs)**

***Table1.1 PEOs of B.E. CSE***

| B.E. Computer Science and Engineering programme will prepare its graduates to | |
| --- | --- |
| **PEO1** | Work productively as successful computer professionals in diverse career paths including supportive and leadership roles on multidisciplinary teams or be active in higher studies. |
| **PEO2** | Communicate effectively, recognize and incorporate societal needs and constraints in their professional endeavors, and practice their profession with high regard to ethical responsibilities. |
| **PEO3** | Engage in life-long learning and to remain current in their profession to foster personal and organizational growth. |

**List of Experiments:**

1. Implement state space search using the following algorithms

i. Hill climbing algorithm

ii. A\* algorithm

2. Adversarial search and game playing

3. Creating rule base and infer the proof using First order Predicate

4. Solving n-Queens problem

5. Solving travelling salesman problem

6. Develop multi agent system for a real time problem

7. Information retrieval using semantic search

8. Designing a chat bot application

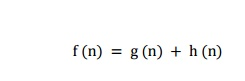
**Ex. No :1 A\* search**

**AIM:**

To find shortest path optimally in a path finding problem using heuristic function

**Description:**

A\* is a combination of Dijkstra’s algorithm and Best first search algorithm. It can be used to solve many kinds of problems such as Path finding, maze problem, games and web-based maps. Fig. 1 shows one such graph with 6 vertices, connected with edges, in which actual cost to reach any node from starting node and heuristic function values of each node. A\* search finds the shortest path in a graph through a search space to goal state using heuristic function. This technique finds minimal cost solutions and is directed to a goal state. In A\*, the \* is written for optimality purpose. The A\* algorithm also finds the lowest cost path between the start and goal state, where changing from one state to another requires some cost. A\* requires heuristic function to evaluate the cost of path that passes through the particular state. This algorithm is complete if the branching factor is finite and every action has fixed cost. It can be defined by following formula.

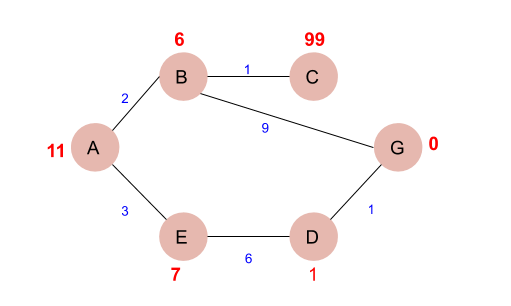


Where

g(n) - The actual path cost from the start state to the current state.

h(n) - The estimated path cost from the current state to goal state.

f(n) - The total actual cost path cost from the start state to the goal state.



**Fig. 1 Graph in which A is the start node and G is the goal node**

A\* maintains two lists- OPEN and CLOSED. OPEN contains those nodes that have been evaluated by the heuristic function but have not been expanded into successors yet. CLOSED contains those nodes that have already been visited.

**Algorithm:**

1. Place the starting node in the OPEN list.
2. Check if the OPEN list is empty or not, if the list is empty then return failure and stops.
3. Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise
4. Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.
5. Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.
6. Return to **Step 2**.

**Exercise:**

Write a JAVA /Python program to implement a path finding problem using A\* search.

The expected output is a optimal path with lowest cost.

**Ex. No :2 Hill climbing search**

**AIM:**

To find the best solution for the problem such as Travelling Sales man.

**Description:**

Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value. It is a technique which is used for optimizing the mathematical problems. One of the widely discussed examples of Hill climbing algorithm is Traveling-salesman Problem in which we need to minimize the distance traveled by the salesman. It is also called greedy local search as it only looks to its good immediate neighbor state and not beyond that. **It only evaluates the neighbor node state at a time and selects the first one which optimizes current cost and set it as a current state**. It only checks it's one successor state, and if it finds better than the current state, then move else be in the same state.

**Algorithm:**

1. Evaluate the initial state, if it is goal state then return success and Stop.
2. Loop Until a solution is found or there is no new operator left to apply.
3. Select and apply an operator to the current state.
4. Check new state:
   1. If it is goal state, then return success and quit.
   2. Else if it is better than the current state then assign new state as a current state.
   3. Else if not better than the current state, then return to step2.
5. Exit.

**Exercise:**

Write a JAVA /Python program to implement Hill climbing for Network flow.

The expected output is best path finding for the problem.

**Ex.No.3 First Order Predicate Logic**

**Aim**

To create rule base and infer the proof using First order Predicate logic.

**Description**

The first order logic assumes that the world contains objects, relations and functions.

**i. Syntax for first order logic:**

* In prepositional logic, every expression is a sentence that represents a fact.
* First order logic includes the sentences along with terms which can represent the objects.
* Constant symbols, variables and function symbols are used to build terms, while quantifiers and predicate symbols are used to build the sentences.

**Syntax:**

| **Constants** | A, B, C..... |
| --- | --- |
| **Functions** | Size, Color |
| **Variable** | x, a |
| **Terms** | Constant, variable or function(Term..) |
| **Predicates** | True, False |
| **Quantifiers** | ∀, ∃ |
| **Atomic sentences** | Predicate, Predicate(Term,…), Term=Term |
| **Sentences** | ¬  Sentence, Sentence ∨ Sentence, Sentence ∧ Sentence, Sentence ⇒ Sentence, Sentence ⇔ Sentence, Quantifier Variable,… Sentence |

**ii. Semantics:**  
  
**Lets understand with an example,**  
Consider the sentence “Elephants are big”.There are many ways to represent this sentence.  
HasSize(Elephant, Big)  
SizeOF(Elephant)= Big  
  
**Lets introduce a new syntax,**  
IsEqual(SizeOf(Elephant, Big), this states that a object Elephant is big, which is a useless fact in any reasoning process about the Elephants in general. So lets represent that all Elephants are big.  
So, we can find**FOL** statement as,

* All things that are Elephants are big.
* For all things x, for which x is a Elephant, x is big.
* For all things x, if x is a Elephant, then x is big.

Finally the **FOL** will be written as.  
∀x Elephant (x) ⇒ Big(x)

Some Examples of FOL using quantifier:

**1. All birds fly.**

In this question the predicate is "**fly(bird)**."

And since there are all birds who fly so it will be represented as follows.  
              **∀x bird(x) →fly(x)**.

**2. Every man respects his parent.**

In this question, the predicate is "**respect(x, y)," where x=man, and y= parent**.  
Since there is every man so will use ∀, and it will be represented as follows:  
              **∀x man(x) → respects (x, parent)**.

**3. Some boys play cricket.**

In this question, the predicate is "**play(x, y)**," where x= boys, and y= game. Since there are some boys so we will use **∃, and it will be represented as**:  
              **∃x boys(x) → play(x, cricket)**.

**4. Not all students like both Mathematics and Science.**

In this question, the predicate is "**like(x, y)," where x= student, and y= subject**.  
Since there are not all students, so we will use **∀ with negation, so** following representation for this:

**¬∀ (x) [ student(x) → like(x, Mathematics) ∧ like(x, Science)].**

**5. Only one student failed in Mathematics.**

In this question, the predicate is "**failed(x, y)," where x= student, and y= subject**.  
Since there is only one student who failed in Mathematics, so we will use following representation for this:

**∃(x) [ student(x) → failed (x, Mathematics) ∧∀ (y) [¬(x==y) ∧ student(y) → ¬failed (x, Mathematics)]**.

**Procedure:**

1. Open the Prolog window and select File->New Option.

2. In the new window, we can write all our rules and save the file name with the ".pl" extension.

3. Compile our ".pl" file by choosing the "compile buffer option.

4. Finally, we can check our rules on the console or command prompt.

**Exercise:**

Write a Prolog program to create a rule base to solve the Monkey Banana problem using First order logic.

**Ex. No : 4 N QUEENS PROBLEM**

**AIM:**

To place all the N Queen in an N×N chessboard such that no two queens attack each other.

**Description:**

It involves placing N chess queens on an N×Nchessboard so that no two queens threaten each other. Fig. 1 depicts three solutions on a standard 8×8board and one on a larger 10 ×10 one well known N-queens puzzle [1] as a benchmark. It involves placing N chess queens on an N×Nchessboard so that no two queens threaten each other. Fig. 1 depicts three solutions on a standard 8×8board and one on a larger 10 ×10 one N-queens puzzle is a well-known and benchmark problem. It involves placing N chess queens on an N×N chessboard so that no two queens threaten each other. Fig. 1 depicts a solution on a standard 8×8 board. The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes then we backtrack and return false.

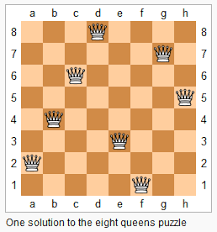


Fig. 1 One Solution to the eight queen puzzle

**Algorithm:**

1. Start in the leftmost column
2. If all queens are placed return true
3. Try all rows in the current column. Do following for every tried row.

a) If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.

b) If placing queen in [row, column] leads to a solution then return true.

c) If placing queen doesn't lead to a solution then unmark this [row, column] (Backtrack) and go to step (a) to try other rows.

1. If all rows have been tried and nothing worked, return false to trigger Backtracking.

**Exercise:**

Write a JAVA /Python program to implement a 4 queen problem using backtracking. The expected output is a binary matrix that has 1s for the blocks where queens are placed.

**Ex.No: 5 ADVERSARIAL SEARCHES**

**AIM:**

**To implement Adversarial Search for Single Agent / Multi agent games.**

**Description:**

Adversarial search problems are which the agents’ goals are in conflict, often known as games. In AI, the most common games are of a rather specialized kind—what game theorists call deterministic, turn-taking, two-player, zero-sum games of perfect information (such as chess). Games exhibit deterministic, fully observable environments in which two agents act alternately and in which the utility values at the end of the game are always equal and opposite. For example, if one player wins a game of chess, the other player necessarily loses. It is this opposition between the agents’ utility functions that makes the situation adversarial.

**Algorithm:**

A game can be formally defined as a kind of search problem with the following elements:

1. S0: The initial state, which specifies how the game is set up at the start.
2. PLAYER(s): Defines which player has the move in a state.
3. ACTIONS(s): Returns the set of legal moves in a state.
4. RESULT(s, a): The transition model, which defines the result of a move.
5. TERMINAL-TEST(s): A terminal test, which is true when the game is over and false otherwise. States where the game has ended are called terminal states.
6. UTILITY(s, p): A utility function (also called an objective function or payoff function)

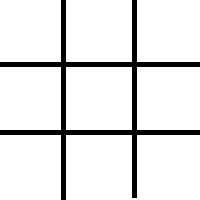
Then create a Game tree, where the nodes are game states and the edges are moves.

Finally decide upon an optimal solution to the game, so as to decide which player has won or to give the score.

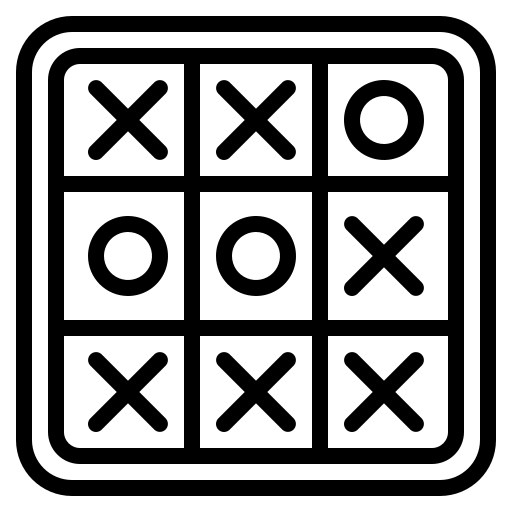
**Exercise:**

Design a Game tree and an Adversarial Search strategy.

The initial state is:



And, one of the Final states may be:



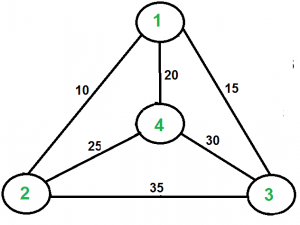
**Ex.No: 6 TRAVELLING SALES PERSON PROBLEM**

**AIM:**

To implement any scheme to find the optimal solution for the Traveling Sales Person problem

**Description:**

The traveling salesman problem (TSP) is an algorithmic problem tasked with finding the shortest route between a set of points and locations that must be visited. In the problem statement, the points are the cities a salesperson might visit. The salesman‘s goal is to keep both the travel costs and the distance traveled as low as possible.



For example, consider the graph shown above. A TSP tour in the graph is 1-2-4-3-1. The cost of the tour is 10+25+30+15 which is 80.

**Algorithm:**

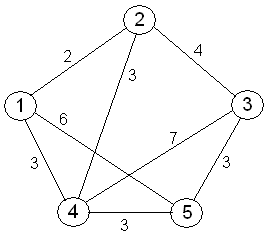
**Naive Solution:**

1. Consider city 1 as the starting and ending point. Since the route is cyclic, we can consider any point as a starting point.
2. Generate all (n-1)! permutations of cities.
3. Calculate the cost of every permutation and keep track of the minimum cost permutation.
4. Return the permutation with minimum cost.

**Dynamic Programming:**

1. Given set of vertices be {1, 2, 3, 4,….n}.
2. Consider vertices 1 as starting and ending point of output.
3. For every other vertex i (other than 1), we find the minimum cost path with 1 as the starting point,’ i’ as the ending point and all vertices appearing exactly once.
4. Let the cost of this path be cost(i), the cost of corresponding Cycle would be cost(i) + dist(i, 1) where dist(i, 1) is the distance from i to 1.
5. Finally, we return the minimum of all [cost(i) + dist(i, 1)] values.**Exercise:**

Given a set of cities and distance between every pair of cities, find the shortest possible route that visits every city exactly once and returns to the starting point. Write a JAVA /Python program to implement the same, assume vertex 1 as a starting city from where sales person starts his travelling.



**Ex.No.7 Information retrieval using semantic search**

**Aim:**

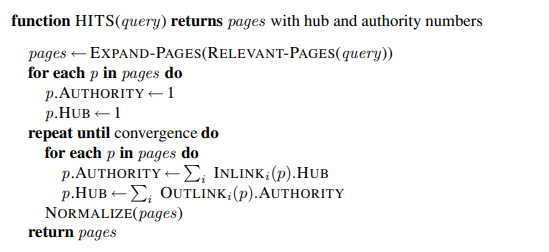
To retrieve information from a Multimedia Document using Semantic search.

**Description:**

Information retrieval is the task of finding documents that are relevant to a user’s need for information. The best-known examples of information retrieval systems are search engines on the World Wide Web. An information retrieval (IR) system can be characterized by:

1. A corpus of documents. Each system must decide what it wants to treat as a document: a paragraph, a page, or a multipage text.
2. Queries posed in a query language. A query specifies what the user wants to know. The query language can be just a list of words, such as [AI book]; or it can specify a phrase of words that must be adjacent, as in [“AI book”]; it can contain Boolean operators as in [AI AND book]; it can include non-Boolean operators such as [AI NEAR book] or [AI book site:www.aaai.org].
3. A result set. This is the subset of documents that the IR system judges to be relevant to the query. By relevant, we mean likely to be of use to the person who posed the query, for the particular information need expressed in the query.
4. A presentation of the result set. This can be as simple as a ranked list of document titles or as complex as a rotating color map of the result set projected onto a threedimensional space, rendered as a two-dimensional display.

**The PageRank algorithm:**



**Exercise:**

Design a system to do segmentation of words without spaces. Given a string, such as the URL “thelongestlistofthelongeststuffatthelongestdomainnameatlonglast.com,” return a list of component words: [“the,” “longest,” “list,” ...]. This task is useful for parsing URLs, for spelling correction when words runtogether, and for languages such as Chinese that do not have spaces between words. It can be solved with a unigram or bigram word model and a dynamic programming algorithm similar to the Viterbi algorithm.

**Ex.No.8 Chatbot**

**Aim**

To create a chatbot to conduct an on-line chat conversation via text or text-to-speech.

**Description**

A chatbot is a computer program that uses [artificial intelligence](https://www.ibm.com/cloud/learn/what-is-artificial-intelligence) (AI) and [natural language processing](https://www.ibm.com/cloud/learn/natural-language-processing) (NLP) to understand customer questions and automate responses to them, simulating human conversation.

**How are Chatbots Trained?**

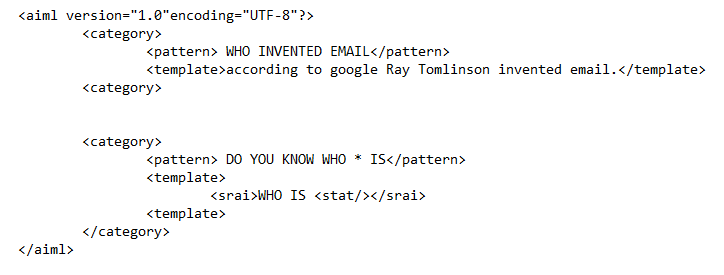
Training a chatbot occurs at a considerably faster and larger scale than human education. While normal customer service representatives are given a manual instruction which they must be thorough with, a customer support chatbot is nourished with a large number of conversation logs, and from those logs, the chatbot can understand what type of question needs, what kind of answers.

**Architecture & Work Methods of Chatbots**

The Chatbots work based on three classification methods:  
**1.Pattern Matches:**

Bots utilize pattern matches to group the text and it produces an appropriate response from the clients. “[Artificial Intelligence](https://mindmajix.com/artificial-neural-network-and-how-it-works) Markup Language (AIML), is a standard structured model of these Patterns.

A simple example of Pattern matching is;

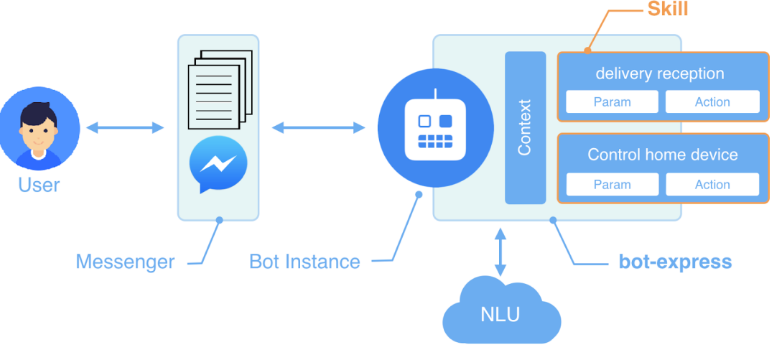


Then the machine gives the following output:

**Human:** Who invented the email?

**Robot:** According to Google, Ray Tomlinson invented email.

The Chatbot knows the appropriate answer because her or his name is in the related pattern. Similarly, the chatbots react to anything relating it to the correlate patterns. But it can’t go past the related pattern. To take it to a progressive stage, algorithms can help.  
For every sort of question, a remarkable pattern must be accessible in the database to give a reasonable response. With a number of pattern combinations, it makes a hierarchical structure. We utilize algorithms to lessen the classifiers and produce the more reasonable structure.  
**2. Natural Language Understanding (NLU)**



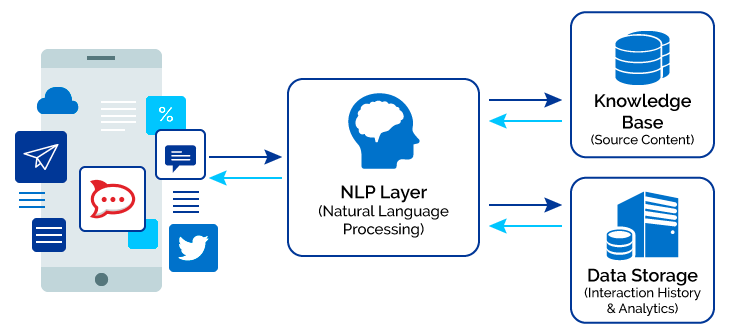
**Fig.1** Architecture of NLU

This NLU has 3 specific concepts as follows:

**Entities:**This essentially represents an idea to your chatbot. For example, it may be a payment system in your E-commerce chatbot.

**Context:** When a natural language understanding algorithm examines a sentence, it doesn’t have the historical backdrop of the user’s text conversation. This implies that, if it gets a response to a question it has been recently asked, it won’t recall the inquiry. So, the phases during the conversation of chat are separately stored. It can either be banners like “Ordering Pizza”. Or could include other parameters like “Domino’s: Restaurant”. With context, you can easily relate expectations with the necessity of comprehending the last question.

**Expectations:** This is what a chatbot must fulfill when the customer says sends an inquiry. Which can be the same for different inquiries. For example, the goal triggered for, “I want to purchase a white pair of shoes”, and “Do you have white shoes? I want to purchase them” or “show me a white pair of shoes”, is the same: a list of shops selling white shoes. Hence, all user typing text show a single command which is the identifying tag; white shoes.  
**3. Natural Language Processing (NLP)**



**Fig.2** Architecture of NLP

Natural Language Processing Chatbots finds a way to convert the user’s speech or text into structured data. Which is then utilized to choose a relevant answer? Natural Language Processing includes the following steps;

1. Tokenization: The NLP separates a series of words into tokens or pieces that are linguistically representative, with a different value in the application.
2. Sentiment Analysis: It will study and learn the user’s experience, and transfer the inquiry to a human when necessary
3. Normalization: This program model processes the text to find out the typographical errors and common spelling mistakes that might alter the intended meaning of the user’s request.
4. Named Entity Recognition: The program model of chatbot looks for different categories of words, similar to the name of the particular product, the user’s address or name, whichever information is required.
5. Dependency Parsing: The Chatbot searches for the subjects, verbs, objects, common phrases and nouns in the user’s text to discover related phrases that what users want to convey.

**Procedure**

[Step 1: Identify the purpose of your chatbot](https://www.tidio.com/blog/how-to-create-a-chatbot-for-a-website/#give-your-chatbot-a-purpose)

[Step 2: Decide where you want it to appear](https://www.tidio.com/blog/how-to-create-a-chatbot-for-a-website/#decide-where-you-want-it-to-appear)

[Step 3: Choose the chatbot platform](https://www.tidio.com/blog/how-to-create-a-chatbot-for-a-website/#choose-the-chatbot-platform)

[Step 4: Design the chatbot conversation in a chatbot editor](https://www.tidio.com/blog/how-to-create-a-chatbot-for-a-website/#design-the-chatbot-conversation-in-a-chatbot-editor)

[Step 5: Test your chatbot](https://www.tidio.com/blog/how-to-create-a-chatbot-for-a-website/#test-your-chatbot)

[Step 6: Train your chatbot](https://www.tidio.com/blog/how-to-create-a-chatbot-for-a-website/#train-your-chatbot)

[Step 7: Collect feedback from users](https://www.tidio.com/blog/how-to-create-a-chatbot-for-a-website/#collect-feedback-from-users)

[Step 8: Monitor chatbot analytics to improve it](https://www.tidio.com/blog/how-to-create-a-chatbot-for-a-website/#monitor-chatbot-analytics-to-improve-it)

**Exercise:**

Create a chatbot to conduct an on-line chat conversation using IBM Watson/ Rasa / Azure/ Python.