Python Chatbot one day Bootcamp

LITHAN

Vision
Deliver Future Ready Talents

Mission
Provide On-Demand Learning and Mentoring

Values
Learn, Innovate and Grow

AM Session

IU#	Instructional Unit Name (30min)
01	Basics Introduction to Python
02	Functions in Python
Demo#	**Demo Name(30min)**
01	Basic Python Program in Jupyter Notebook
IU#	**Instructional Unit Name (1hr)**
03	Conditionals in Python
04	Nesting and Loops in Python
05	Introduction to chatbot and its types/advantages
Demo#	**Demo Name(1hr)**
02	Chatbot Program using conditionals and Functions in Python
03	Chatbot Program using conditionals and Looping in Python

Basics Introduction to Python

Basics Introduction to Python

- Getting started with Python in Jupyter Notebooks
- Python 3 in Jupyter notebooks**
  - `print()`
  - comments
  - data types basics
  - Variable
  - Using code to write "Hello World!" on the screen is the traditional first program when learning a new language in computer science
  - print("Hello World!")

```
print("Hello World!")
```

```
Hello World!
```

Basics Introduction to Python – Strings

- WHAT IS A STRING? Strings were used in "Hello World!" print function examples. Strings are sets of characters. In Python strings are in double or single quotes like "Hello World!" or 'after edit, save!'
- A String is a common type of data, used in programming, consisting of a sequence of 1 or more characters.
  - A String is a sequence of characters (aka: a string of characters)
  - Character examples: A, B, C, a, b, c, 1, 2, 3, !, &
  - A String in python is contained in quotes, single(') or double(")
  - String examples: "ABC", 'Joana Dias', 'I have 2 pet cats.', "item #3 cost $3.29 USD"
  - Note: the quotes containing a string must come in matched pairs
  - "Hello" or 'Hello' are correctly formatted strings
  - "Hello' is incorrectly formatted starting with double " and ending with single ' python needs a matching quote to know where the string ends

Basics Introduction to Python – Strings

- ```
  # [ ] enter a string in the print() function using single quotes
  print('strings go in single')
  # [ ] enter a string in the print() function using double quotes
  print("or double quotes")
  ```

  ```
  strings go in single
  or double quotes
  ```
  ```

Basics Introduction to Python – Integers

- WHAT IS A INTEGER?
  - whole numbers such as -2, -1, 0, 1, 2, 3 are Integers Integers are not placed in quotes A String can contain any character, this includes letters, spaces, punctuation, number digits and formatting. In a string with digits ("123")
  - the digits are text images rather than representations of numeric values. remember a line of python code starting with the pound or hash symbol
  - l (#) indicates a comment comments are for humans to read and are not run by computers

```
# printing an Integer with python
print(299)
# printing a string made of Integer (number) characters with python
print("2017")
```

```
299
2017
```

Basics Introduction to Python – Variables

- WHAT IS A VARIABLE ?
  - Variables are named containers Computer programs often create storage for things that are used in repeated processing like item_price, student_name, stock_symbol. In python a variable is a type of object that can be addressed by name to access the assigned contents. Name a variable staring with a letter or underscore "_”
  - There are different styles for creating variables this course uses lowercase descriptive names connected by underscores:
  - item_price student_name stock_symbol
  - descriptive names reduce the need for comments in the code and make it easier to share code or read code written long ago
  - Variables can hold strings once a variable is assigned: current_msg = "I am a string"

Basics Introduction to Python – Variables

- Variable reassignment

- We can reassign a variable, changing the contents. current_msg = "new current message" Variables can be used in place of literals

- If we initialize current_msg = "new current message" then the literal string "new current message" and the variable current_msg are the same when used in code

Basics Introduction to Python – Variables

- **initialize the variable**
  - current_msg = "I am a string"
- **print literal string**
  - print("I am a string")
- **print variable string**
  - print(current_msg)
- **assign a new string to the current_msg**
  - current_msg = "Run this cell using Ctrl+Enter"
  - print(current_msg)

Basics Introduction to Python – Variables

```
# initialize the variable
current_msg = "I am a string"

# print literal string
print("I am a string")

# print variable string
print(current_msg)
 # assign a new string to the current_msg
current_msg = "Run this cell using Ctrl+Enter"

print(current_msg)
```

```
I am a string
I am a string
Run this cell using Ctrl+Enter
```

Basics Introduction to Python – Data Types in Variables

- variables can be initialized with different data types. Below we see item_price is a decimal (aka - float) and student_name is string
- item_price = 10.25 #item_price initialized as a numeric value (no quotes) student_name ="Dias, Joana" #student_name initialized as a string license_plate = "123A" #license_plate initialized as a string Variables can start initialized as an Integer number data type

```
name = "Joana Dias"
print(name)
test_value = 22
print(test_value)
test_value = "Joana"
print(test_value)
```

```
Joana Dias
22
Joana
```

Basics Introduction to Python – using Type function

- type() returns the data type of Python objects str, int, float What does using type() reveal?
- str: when type() returns str that means it has evaluated a string characters (numbers, letters, punctuation...) in quotes
- int: when type() returns int that means it has evaluated an Integer (+/- whole numbers)
- float: when type() returns float that means it has evaluated decimal numbers (e.g. 3.33, 0.01, 9.9999 and 3.0

```
type("Hello World!")
```

```
str
```

```
type(501)
```

```
int
```

```
type(8.33333)
```

```
float
```

```
student_name = "Colette Browning"
type(student_name)
```

```
str
```

Basics Introduction to Python – addition in PRINT ()

- Use print() to show the results of multiple lines of output
- **assigning variables & using addition**
  - add_two = 34 + 16
  - first_name = "Alton"
  - greeting = "Happy Birthday " + first_name
  - print(add_two)
  - print(greeting)

```
add_two = 34 + 16
first_name = "Alton"
greeting = "Happy Birthday " + first_name
print(add_two)
print(greeting)
```

```
50
Happy Birthday Alton
```

Basics Introduction to Python – addition in PRINT ()

- Use print() to show the results of multiple lines of output
- **Integer addition in variables and in a print function**
  - int_sum = 6 + 7
  - print(int_sum)
  - print(11 + 15) print()

```
int_sum = 6 + 7
print(int_sum)
print(11 + 15)
print()
```

```
13
26
```

- **string addition in variables and in print()function**
  - hat_msg = "I do not wear " + "a hat”
  - print(hat_msg)
  - print("at " + "dinner")

```
hat_msg = "I do not wear " + "a hat"
print(hat_msg)
print("at " + "dinner")
```

```
I do not wear a hat
at dinner
```

Basics Introduction to Python – user input ()

- get information from users with input()
- the input() function prompts the user to supply data returning that data as a string
- **enter a small integer in the text box**
  - print("enter a small int: ")
  - small_int = input()
  - print("small int: ")
  - print(small_int)

```
print("enter a small int: ")
small_int = input()
print("small int: ")
print(small_int)
```

```
enter a small int:
1
small int:
1
```

Basics Introduction to Python – user input ()

- USER PROMPTS USING INPUT()
- the input() function has an optional string argument which displays the string intended to inform a user what to enter input() works similar to print() in the way it displays arguments as output
- **enter a small integer in the text box**
  - student_name = input("enter the student name: ")
    print("Hi " + student_name)

```
student_name = input("enter the student name: ")
print("Hi " + student_name)
```

```
enter the student name: John
Hi John
```

Basics Introduction to Python – print formatting()

- Python provides several methods of formatting strings in the print() function beyond string addition The print() function supports using commas to combine strings for output by comma separating strings print() will output each string separated with a space by default comma formatted print()

```
name = "John"

print("Hello to",name,"who is from the city")
```

```
Hello to John who is from the city
```

Functions in Python

Functions in Python – Simple Functions

- CREATE A SIMPLE FUNCTION
  - Creating user defined functions is at the core of computer programming.
  - Functions enable code reuse and make code easier to develop and maintain.
- Basics of a user defined function
  - define a function with def use indentation (4 spaces)
    - define parameters optional parameters return values (or none)
  - Function scope (basics defaults) def some_function():

Functions in Python – Simple Functions

- use a function name that starts with a letter or underscore (usually a lower-case letter)
- function names can contain letters, numbers or underscores parenthesis () follow the function name a colon :
- follows the parenthesis the code for the function is indented under the function definition (use 4 spaces for this course)
- Eg : def some_function():

Functions in Python – Simple Functions – Creation Examples

- **defines a function named say_hi**
  - def say_hi():
  - print("Hello there!")
  - print("goodbye")
- **define three_three**
  - def three_three():
  - print(33)

```
def say_hi():
    print("Hello there!")
    print("goodbye")
# end of indentation ends the function

# define three_three
def three_three():
    print(33)
```

Functions in Python – Simple Functions – Calling

- Call a simple function using the function name followed by parentheses.
- For instance, calling print function is print()

```
def say_hi():
    print("Hello there!")
    print("goodbye")
# end of indentation ends the function

# define three_three
def three_three():
    print(33)

# calling the functions
say_hi()
print()
three_three()
```

```
Hello there!
goodbye

33
```

Functions in Python –Functions with arguments/Parameters

- print() and type() are examples of built-in functions that have Parameters defined
- type() has a parameter for a Python Object and sends back the type of the object
- an Argument is a value given for a parameter when calling a function
- type is called providing an Argument - in this case the string "Hello" type("Hello") DEFINING FUNCTION PARAMETERS

Functions in Python –Functions with arguments/Parameters

```
# yell_this() yells the string Argument provided
def yell_this(phrase):
    print(phrase.upper() + "!")

# call function with a string
yell_this("It is time to learn Python ")
# use a default argument
def say_this(phrase = "Hi"):
    print(phrase)

say_this()
say_this("Bye")
```

```
IT IS TIME TO LEARN PYTHON !
Hi
Bye
```

Functions in Python –Functions with return value

- type() returns an object type type() can be called with a float the return value can be stored in a variable object_type = type(2.33)
- CREATING A FUNCTION WITH A RETURN VALUE return keyword in a function returns a value after exiting the function
    - def msg_double(phrase):
    - double = phrase + " " + phrase
    - return double
- Calling the function
    - msg_2x = msg_double("let's go to Lithan")
    - print(msg_2x)

```
def msg_double(phrase):
    double = phrase + " " + phrase
    return double
msg_2x = msg_double("let's go to Lithan ")
print(msg_2x)
```

```
let's go to Lithan  let's go to Lithan
```

Functions in Python –Functions with return value

- type() returns an object type type() can be called with a float the return value can be stored in a variable object_type = type(2.33)
- CREATING A FUNCTION WITH A RETURN VALUE return keyword in a function returns a value after exiting the function
    - def msg_double(phrase):
    - double = phrase + " " + phrase
    - return double
- Calling the function
    - msg_2x = msg_double("let's go to Lithan")
    - print(msg_2x)

```
def msg_double(phrase):
    double = phrase + " " + phrase
    return double
msg_2x = msg_double("let's go to Lithan ")
print(msg_2x)
```

```
let's go to Lithan  let's go to Lithan
```

Functions in Python –Functions with return value

```
def make_schedule(period1, period2):
    schedule = ("[1st] " + period1.title() + ", [2nd] " + period2.title())
    return schedule

student_schedule = make_schedule("Python programming", "Data Science ")
print("SCHEDULE:", student_schedule)
```

```
SCHEDULE: [1st] Python Programming, [2nd] Data Science
```

Demo : Execute all the programs from the cloned project – in Basics

Conditionals in Python

Conditionals in Python

- CONDITIONALS USE TRUE OR FALSE
- if True: print("True means do something") else: print("Not True means do something else") hot_tea = True

```
if True:
    print("True means do something")
else:
    print("Not True means do something else")
hot_tea = True

if hot_tea:
    print("enjoy some hot tea!")
else:
    print("enjoy some tea, and perhaps try hot tea next time.")
```

Conditionals in Python

- If student_name
  - .isalpha():
  - .isalnum()
  - .istitle()
  - .isdigit()
  - .islower()
  - .startswith()

```
favorite_book = input("Enter the title of a favorite book: ")

if favorite_book.istitle():
    print(favorite_book, "- nice capitalization in that title!")
else:
    print(favorite_book, "- consider capitalization throughout for book titles.")
```

Conditionals in Python – Multiple conditions

- Review: if and else
- if means "if a condition exists then do some task." if is usually followed by else
- else means "or else after we have tested if, then do an alternative task" When there is a need to test for multiple conditions there is elif
- elif statement follows if, and means "else, if " another condition exists do something else
- elif can be used many times
- else is used after the last test condition (if or elif) in pseudo code

Conditionals in Python – Multiple conditions

- Example
- If it is raining bring an umbrella or
- Else If (elif) it is snowing bring a warm coat or
- Else go as usual
- Like else, the elif only executes when the previous if conditional is False

Conditionals in Python – Multiple conditions

```
weather = input("Enter weather (sunny, rainy, snowy): ")

if weather.lower() == "sunny":
    print("Wear a t-shirt")
elif weather.lower() == "rainy":
    print("Bring an umbrella and boots")
elif weather.lower() == "snowy":
    print("Wear a warm coat and hat")
else:
    print("Sorry, not sure what to suggest for", weather)
```

```
Enter weather (sunny, rainy, snowy): sunny
Wear a t-shirt

Enter weather (sunny, rainy, snowy): rainy
Bring an umbrella and boots
```

Conditionals in Python – Multiple conditions

- Example
- If it is raining bring an umbrella or
- Else If (elif) it is snowing bring a warm coat or
- Else go as usual
- Like else, the elif only executes when the previous if conditional is False

Loops in Python

Loops in Python –

- while True:
  - is known as the forever loop because it ...loops forever
- Using the while True: statement results in a loop that continues to run forever ...or, until the loop is interrupted, such as with a break statement
  - BREAK
- in a while loop, causes code flow to exit the loop
- a conditional can implement break to exit a while True loop
-

Loops in Python –

```
while True:
    weather = input("Enter weather (sunny, rainy, snowy, or quit): ")
    print()

    if weather.lower() == "sunny":
        print("Wear a t-shirt and sunscreen")
        break
    elif weather.lower() == "rainy":
        print("Bring an umbrella and boots")
        break
    elif weather.lower() == "snowy":
        print("Wear a warm coat and hat")
        break
    elif weather.lower().startswith("q"):
        print('"quit" detected, exiting')
        break
    else:
        print("Sorry, not sure what to suggest for", weather +"\n")
```

```
Enter weather (sunny, rainy, snowy, or quit): sunnuy

Sorry, not sure what to suggest for sunnuy

Enter weather (sunny, rainy, snowy, or quit): sunny

Wear a t-shirt and sunscreen
```

Introduction to Chatbot

What is Chatbot

- Chatbots are the innovation done by intelligent humans to serve humanity intelligently.
- It’s the most advanced & simplest way humans can interact with computers to get their things done.
  - As per Wiki: A chatbot (also known as a smartbot, conversational bot, chatterbot, interactive agent, conversational interface, Conversational AI, talkbot or artificial conversational entity) is a computer program or an artificial intelligence which conducts a conversation via auditory or textual methods.

What is Chatbot

- An artificial intelligence developed to collect or provide online information to customers or learners interactively. Artificial intelligence is changing the way we search and get things done and Chatbots are the real example of human aspiration to get rid of doing things which doesn’t excites them much.
-

How Chatbot Works?

- **What is the technology empowering Them ?**
  - Most of the chatbots are a kind of messaging interface where instead of humans answering to your messages bots are responding. fundamentally they may look like another app. But the catch is that UI layer where you interact.
  - The conversation humans have with bots is powered by ML algorithms which breaks down your messages into human understandable natural languages using NLP techniques and responds to your queries similar to what you can expect from any human on the other side .
  - They usually rely on machine learning, especially on NLP.
- **Examples**
  - Apple’s Siri ,Amazon’s Alexa ,Google Assistant ,Microsoft’s Cortana

Simple Chatbot Program using conditionals and Functions

```
import random
words = ("hi", "hey","greetings")
resp = ["Good Morning"]

def fun(sent):
    for i in sent.split(' '):
        if i.lower() in words:
            return random.choice(resp)
        else:
            return "i dont know "
while True:
    ip = input("You:")
    bot = fun(ip)
    print("Bot:" + bot)
    if "bye" in ip:
        break
```

```
You:hi
Bot:Good Morning
You:
Bot:i dont know
You:bye
Bot:i dont know
```

Simple Chatbot Program using conditionals and Looping

```
import random
import sys
words = (
         "How are you",
          "What is your name",
          "How is the weather today")
resp = ["im fine thank you ",
         "bot",
         "Sunny"]
```

Simple Chatbot Program using conditionals and Looping

```
def wordslist():
    for i in range(len(words)):
        print(str(i)+":"+words[i])
def checkwordsbyindex():
    wordslist()
    Qid = input("Please select your options")
    Resid = ''
    if "bye" in Qid:
        sys.exit()
    elif int(Qid) < len(words):
        Resid = resp[int(Qid)]
    else:
        Resid = "i dont knwo yet"
    return Resid
while True:
    response  = checkwordsbyindex()
    print("bot reposnse:" + response)
```

Simple Chatbot Program using conditionals and Looping

```
0:How are you
1:What is your name
2:How is the weather today
Please select your options0
bot reposnse:im fine thank you
0:How are you
1:What is your name
2:How is the weather today
Please select your options1
bot reposnse:bot
0:How are you
1:What is your name
2:How is the weather today
Please select your options3
bot reposnse:i dont knwo yet
0:How are you
1:What is your name
2:How is the weather today
```

PM Session

IU#	Instructional Unit Name (1hrs)
01	Understanding Chatbot text analytics
02	Introduction to NLTK
Demo#	**Demo Name(1hr)**
01	Simple pair reflection Chatbot using NLTK
IU#	**Instructional Unit Name (1hrs)**
03	Introduction to AI, ML and DL
04	Chatbot using Machine Learning algorithms
Demo#	**Demo Name(2hr)**
02	Chatbot using NLTK
03	Querying Google In Python for ChatBot

Understanding Chatbot text analytics

- There are broadly two variants of chatbots: Rule-Based and Self-learning.
- **In a Rule-based approach,**
  - a bot answers questions based on some rules on which it is trained on. The rules defined can be very simple to very complex. The bots can handle simple queries but fail to manage complex ones.
- **Self-learning bots**
  - are the ones that use some Machine Learning-based approaches and are definitely more efficient than rule-based bots. These bots can be of further two types: Retrieval Based or Generative

Understanding Chatbot text analytics

- In retrieval-based models
  - a chatbot uses some heuristic to select a response from a library of predefined responses.
  - The chatbot uses the message and context of the conversation for selecting the best response from a predefined list of bot messages.
  - The context can include a current position in the dialogue tree, all previous messages in the conversation, previously saved variables (e.g. username).
  - Heuristics for selecting a response can be engineered in many different ways, from rule-based if-else conditional logic to machine learning classifiers. ii) Generative bots can generate the answers and not always replies with one of the answers from a set of answers. This makes them more intelligent as they take word by word from the query and generates the answers.

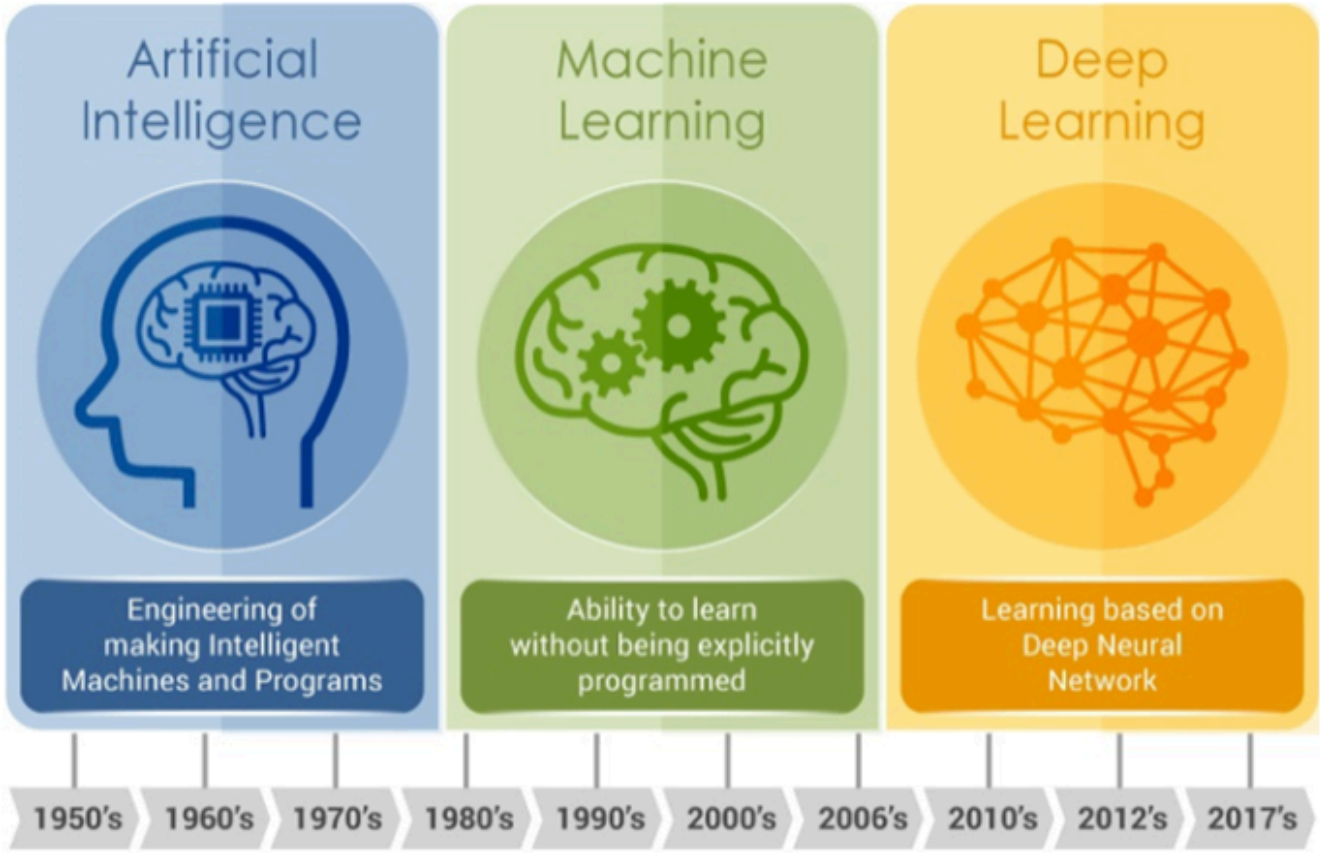
Understanding Chatbot text analytics

- Generative bots
  - can generate the answers and not always replies with one of the answers from a set of answers.
  - This makes them more intelligent as they take word by word from the query and generates the answers.

Understanding Chatbot text analytics

- History of chatbots dates back to 1966 when a computer program called ELIZA was invented by Weizenbaum. It imitated the language of a psychotherapist from only 200 lines of code. You can still converse with it here: Eliza.
- On similar lines let's create a very basic chatbot utlising the Python's NLTK library.It's a very simple bot with hardly any cognitive skills,but still a good way to get into NLP and get to know about chatbot

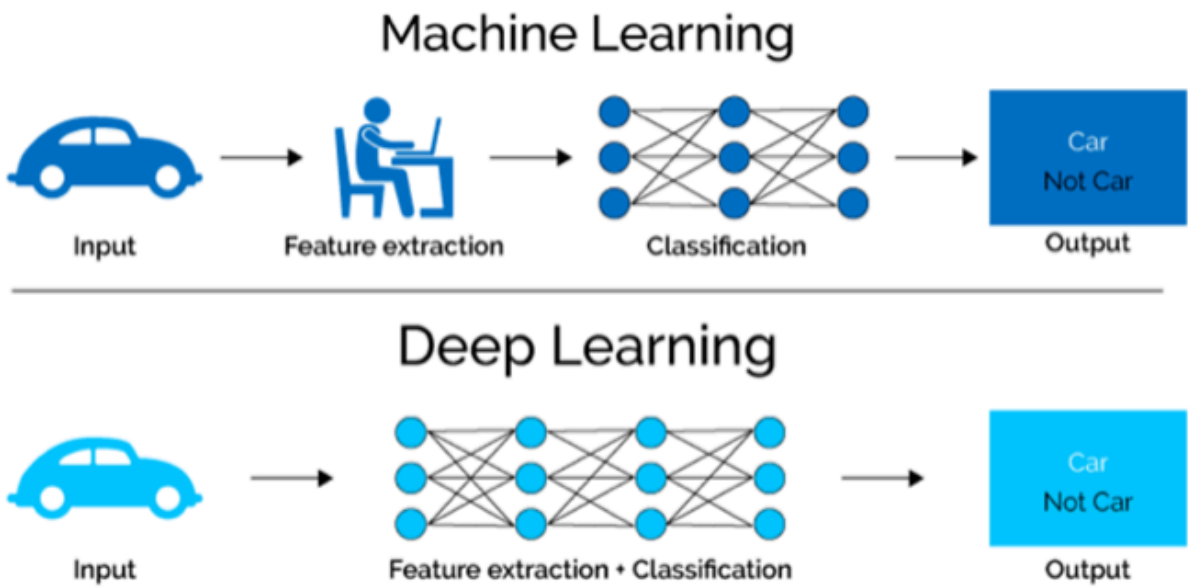
Understanding AI, ML, DL



Evolution of AI — Source: https://www.embedded-vision.com/

- AI is really a broad term and somewhat this also causes every company to claim their product has AI
- Then ML is a subset of AI, consists of the more advanced techniques and models that enable computers to figure things out from the data and deliver AI applications.

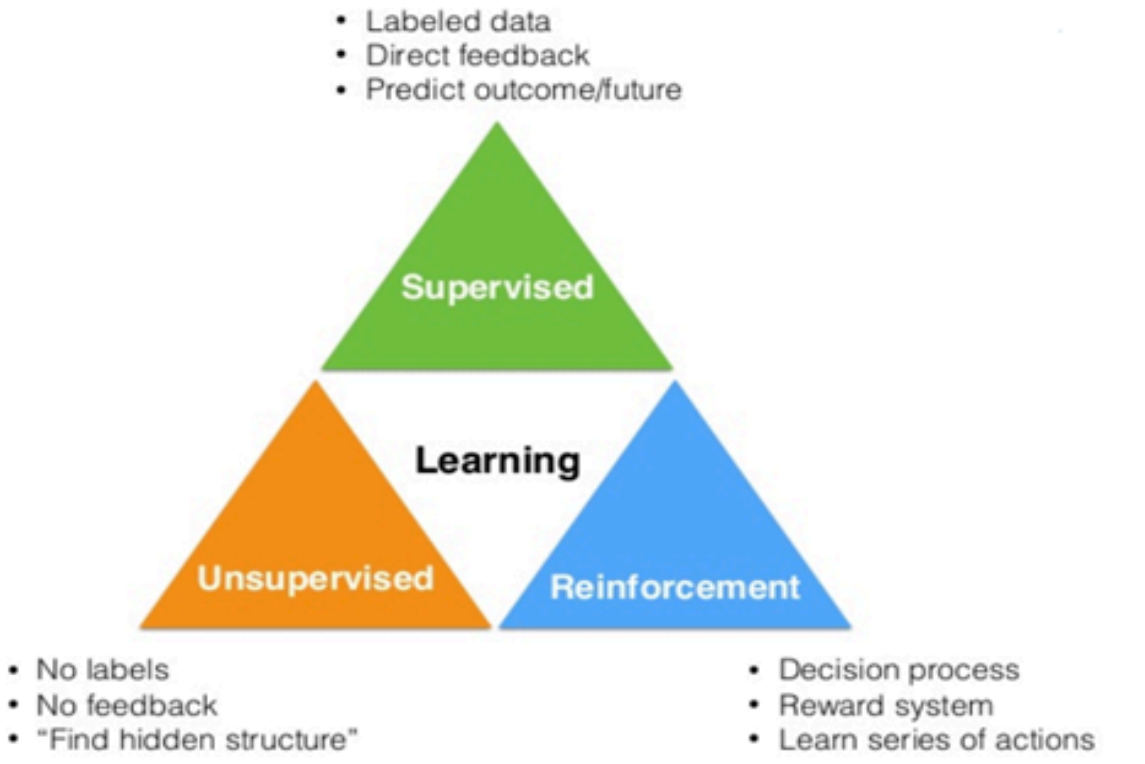
Understanding AI, ML, DL



Source: https://www.xenonstack.com/blog/data-science/log-analytics-deep-machine-learning-ai/

- DL is a newer area of ML that that uses multi-layered artificial neural networks to deliver high accuracy in tasks such as object detection, speech recognition, language translation and other recent breakthroughs that you hear in the news.

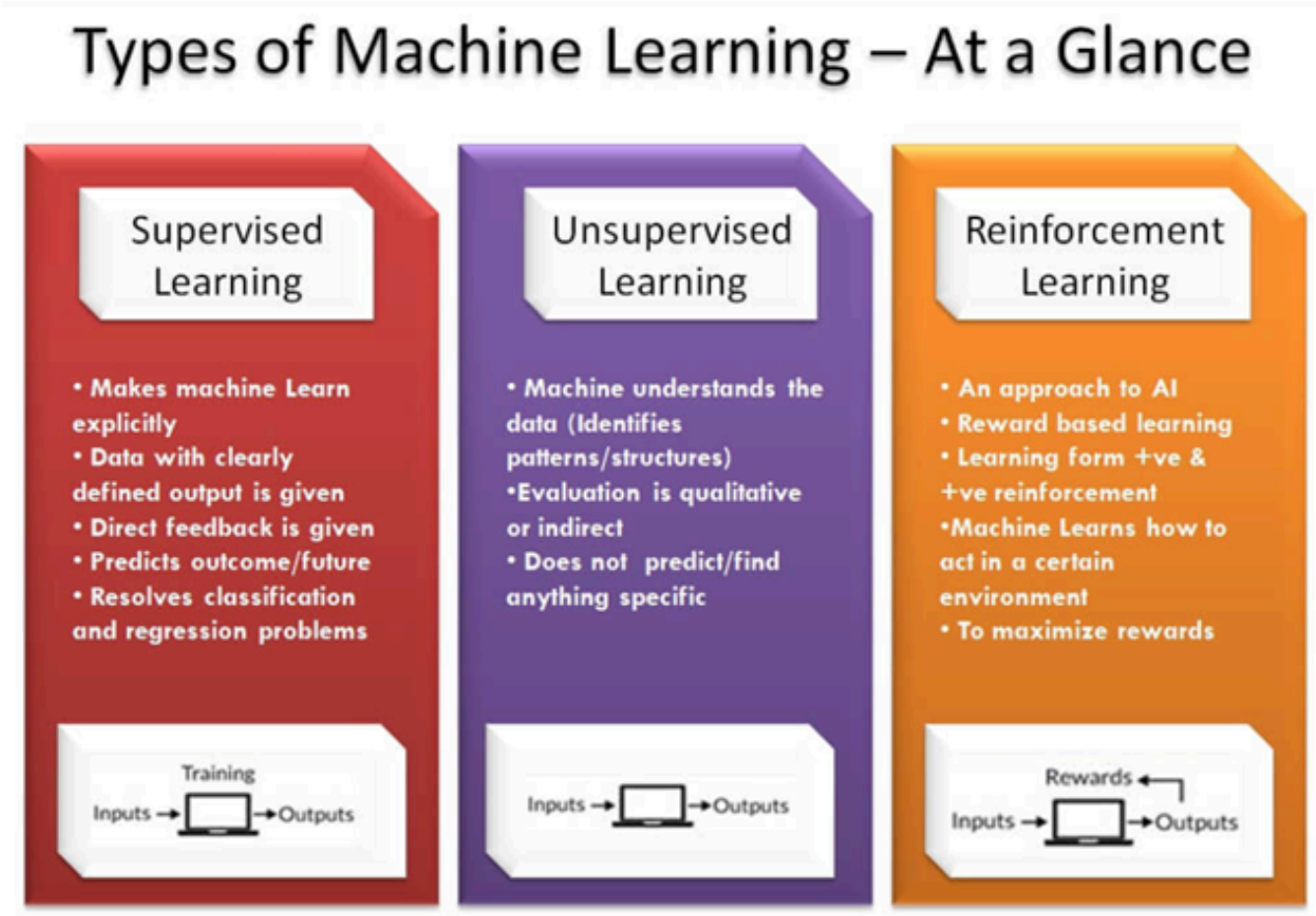
Understanding AI, ML, DL



Source: https://www.slideshare.net/SebastianRaschka/nextgen-talk-022015/8-Learning_Labeled_data_Direct_feedback

- In Machine Learning there are different models that generally fall into 3 different categories:
  (1)**Supervised Learning,**
  (2) Unsupervised Learning
  (3) Reinforcement Learning.

Understanding AI, ML, DL



Source: https://www.newtechdojo.com/list-machine-learning-algorithms/

Understanding DL - Example - NLP & NLTK

- **NLP**
  - The field of study that focuses on the interactions between human language and computers is called Natural Language Processing, or NLP for short. It sits at the intersection of computer science, artificial intelligence, and computational linguistics
  - NLP is a way for computers to analyze, understand, and derive meaning from human language in a smart and useful way. By utilizing NLP, developers can organize and structure knowledge to perform tasks such as automatic summarization, translation, named entity recognition, relationship extraction, sentiment analysis, speech recognition, and topic segmentation.

Understanding NLP & NLTK

- **NLTK**
  - NLTK(Natural Language Toolkit) is a leading platform for building Python programs to work with human language data.
    - It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries.
    - NLTK has been called “a wonderful tool for teaching and working in, computational linguistics using Python,” and “an amazing library to play with natural language.”

Understanding NLP & NLTK

- Downloading and installing NLTK Install NLTK:
  - run pip install nltk from your terminal
  - Test installation: run python then type import nltk

- import NLTK and run nltk.download().

Installing NLTK Packages

```
import nltk
from nltk.stem import WordNetLemmatizer
nltk.download('popular', quiet=True) # for downloading packages
#nltk.download('punkt') # first-time use only
#nltk.download('wordnet') # first-time use only
```

Text Pre- Processing with NLTK

- The main issue with text data is that it is all in text format (strings).
- However, Machine learning algorithms need some sort of numerical feature vector in order to perform the task.
- So before we start with any NLP project we need to pre-process it to make it ideal for work. Basic text pre-processing includes:
- 1. Converting the entire text into uppercase or lowercase, so that the algorithm does not treat the same words in different cases as different

Text Pre- Processing with NLTK

- 1. Converting the entire text into lowercase

Reading in the corpus

For our example,we will be using the Wikipedia page for chatbots as our corp
'chatbot.txt'. However, you can use any corpus of your choice.

```
f=open('chatbot.txt','r',errors = 'ignore')
raw=f.read()
raw = raw.lower()# converts to lowercase
```

Text Pre- Processing with NLTK

- 2. Tokenization: Tokenization is just the term used to describe the process of converting the normal text strings into a list of tokens i.e words that we actually want. Sentence tokenizer can be used to find the list of sentences and Word tokenizer can be used to find the list of words in strings. The NLTK data package includes a pre-trained Punkt tokenizer for English.
- 3. Removing Noise i.e everything that isn't in a standard number or letter.
- 4. Removing Stop words. Sometimes, some extremely common words which would appear to be of little value in helping select documents matching a user need are excluded from the vocabulary entirely. These words are called stop words

Text Pre- Processing with NLTK

- 2. **Tokenization**

Tokenisation ¶

```
|: sent_tokens = nltk.sent_tokenize(raw)# converts to list of sentences
   word_tokens = nltk.word_tokenize(raw)# converts to list of words
```

Text Pre- Processing with NLTK

- 5. Stemming: Stemming is the process of reducing inflected (or sometimes derived) words to their stem, base or root form — generally a written word form. Example if we were to stem the following words: “Stems”, “Stemming”, “Stemmed”, “and Stemtization”, the result would be a single word “stem”.
- 6. Lemmatization: A slight variant of stemming is lemmatization. The major difference between these is, that, stemming can often create non-existent words, whereas lemmas are actual words. So, your root stem, meaning the word you end up with, is not something you can just look up in a dictionary, but you can look up a lemma. Examples of Lemmatization are that “run” is a base form for words like “running” or “ran” or that the word “better” and “good” are in the same lemma so they are considered the same.

Text Pre- Processing with NLTK

- 5. Stemming
- 6. Lemmatization:

Preprocessing ¶

We shall now define a function called LemTokens which will take as input the tokens and return normalized tokens.

```
lemmer = nltk.stem.WordNetLemmatizer()
#WordNet is a semantically-oriented dictionary of English included in NLTK.
def LemTokens(tokens):
    return [lemmer.lemmatize(token) for token in tokens]
remove_punct_dict = dict((ord(punct), None) for punct in string.punctuation)

def LemNormalize(text):
    return LemTokens(nltk.word_tokenize(text.lower().translate(remove_punct_dict)))
```

Text Pre- Processing with NLTK

Keyword matching

Next, we shall define a function for a greeting by the bot i.e if a user's input is a greeting, the bot shall return a greeting response.ELIZA uses a simple keyword matching for greetings. We will utilize the same concept here.

```
GREETING_INPUTS = ("hello", "hi", "greetings", "sup", "what's up","hey",)
GREETING_RESPONSES = ["hi", "hey", "*nods*", "hi there", "hello", "I am glad! You are talking to me"]
def greeting(sentence):

    for word in sentence.split():
        if word.lower() in GREETING_INPUTS:
            return random.choice(GREETING_RESPONSES)
```

Text Pre- Processing with NLTK

- **Bag of Words**
- After the initial preprocessing phase, we need to transform text into a meaningful vector (or array) of numbers. The bag-of-words is a representation of text that describes the occurrence of words within a document. It involves two things:
- A vocabulary of known words.
- A measure of the presence of known words.
-

Text Pre- Processing with NLTK – Bag of words

- **Bag of Words**
- Why is it is called a “bag” of words? That is because any information about the order or structure of words in the document is discarded and the model is only **concerned with whether the known words occur in the document, not where they occur in the document.**
- The intuition behind the Bag of Words is that documents are similar if they have similar content. Also, we can learn something about the meaning of the document from its content alone.
-

Text Pre- Processing with NLTK – Bag of words

- Let's make the bag-of-words model concrete with a worked example.
- **Step 1: Collect Data**
- Below is a snippet of the first few lines of text from the book "A Tale of Two Cities" by Charles Dickens, taken from Project Gutenberg.
  - *It was the best of times,*
    it was the worst of times,
    it was the age of wisdom,
    it was the age of foolishness,
- For this small example, let's treat each line as a separate "document" and the 4 lines as our entire corpus of documents.

Text Pre- Processing with NLTK – Bag of words

- **Step 2: Design the Vocabulary**
- Now we can make a list of all of the words in our model vocabulary.
- The unique words here (ignoring case and punctuation) are:
- “it” , “was” , “the” , “best” , “of” , “times” , “worst” , “age” , “wisdom” , “foolishness”
- That is a vocabulary of 10 words from a corpus containing 24 words.

Text Pre- Processing with NLTK – Bag of words

- **Step 3: Create Document Vectors**
- The next step is to score the words in each document.
- The objective is to turn each document of free text into a vector that we can use as input or output for a machine learning model.
- Because we know the vocabulary has 10 words, we can use a fixed-length document representation of 10, with one position in the vector to score each word.
- The simplest scoring method is to mark the presence of words as a boolean value, 0 for absent, 1 for present.
-

Text Pre- Processing with NLTK – Bag of words

- Using the arbitrary ordering of words listed above in our vocabulary, we can step through the first document (“*It was the best of times*“) and convert it into a binary vector.
- The scoring of the document would look as follows:
- “it” = 1 , “was” = 1 , “the” = 1 , “best” = 1 , “of” = 1 , “times” = 1
- “worst” = 0 , “age” = 0 , “wisdom” = 0 , “foolishness” = 0
- As a binary vector, this would look as follows:
- [1, 1, 1, 1, 1, 1, 0, 0, 0, 0]

Text Pre- Processing with NLTK – Bag of words

- The other three documents would look as follows:
- "it was the worst of times" = [1, 1, 1, 0, 1, 1, 1, 0, 0, 0]
- "it was the age of wisdom" = [1, 1, 1, 0, 1, 0, 0, 1, 1, 0]
- "it was the age of foolishness" = [1, 1, 1, 0, 1, 0, 0, 1, 0, 1]
- All ordering of the words is nominally discarded and we have a consistent way of extracting features from any document in our corpus, ready for use in modeling.
- New documents that overlap with the vocabulary of known words, but may contain words outside of the vocabulary, can still be encoded, where only the occurrence of known words are scored and unknown words are ignored.
- You can see how this might naturally scale to large vocabularies and larger documents

Text Pre- Processing with NLTK – TF-IDF

- **TF-IDF**
- A problem with scoring word frequency is that highly frequent words start to dominate in the document (e.g. larger score), but may not contain as much “informational content” to the model as rarer but perhaps domain specific words.
- One approach is to rescale the frequency of words by how often they appear in all documents, so that the scores for frequent words like “the” that are also frequent across all documents are penalized.
- This approach to scoring is called Term Frequency – Inverse Document Frequency, or TF-IDF for short, where:
- **Term Frequency**: is a scoring of the frequency of the word in the current document.
- **Inverse Document Frequency**: is a scoring of how rare the word is across documents.

Program with NLTK

```
def response(user_response):
    robo_response=''
    sent_tokens.append(user_response)
    TfidfVec = TfidfVectorizer(tokenizer=LemNormalize, stop_words='english')
    tfidf = TfidfVec.fit_transform(sent_tokens)
    vals = cosine_similarity(tfidf[-1], tfidf)
    idx=vals.argsort()[0][-2]
    flat = vals.flatten()
    flat.sort()
    req_tfidf = flat[-2]
    if(req_tfidf==0):
        robo_response=robo_response+"I am sorry! I don't understand you"
        return robo_response
    else:
        robo_response = robo_response+sent_tokens[idx]
        return robo_response
```

Program with NLTK

```
flag=True
print("ROBO: My name is LithanBOT. I will answer your queries about Chatbots. If you want to exit, type Bye!")
while(flag==True):
    user_response = input()
    user_response=user_response.lower()
    if(user_response!='bye'):
        if(user_response=='thanks' or user_response=='thank you' ):
            flag=False
            print("ROBO: You are welcome..")
        else:
            if(greeting(user_response)!=None):
                print("ROBO: "+greeting(user_response))
            else:
                print("ROBO: ",end="")
                print(response(user_response))
                sent_tokens.remove(user_response)
    else:
        flag=False
        print("ROBO: Bye! take care..")
```

Result = Program with NLTK

```
ROBO: My name is LithanBOT. I will answer your queries about Chatbots. If you want to exit, type Bye!
hi
ROBO: hello
any use of chatbot
ROBO: in order to speed up this process, designers can use dedicated chatbot design tools, that allow for immedi
ate preview, team collaboration and video export.an important part of the chatbot design is also centered around
user testing.
```

Simple pair reflection Chatbot using NLTK

```
from __future__ import print_function
from nltk.chat.util import Chat, reflections

pairs = (
     (r'quit',
      ( "Thank you. It was good talking to you.",
        "Good-bye.")),

     (r'Who are you(.*)',
      ( "My name is Alpha.",
        "I'm Alpha.")),

     (r'What is your name(.*)',
      ( "My name is Alpha.",
        "I'm Alpha.")),

     (r'What do you do(.*)',
      ( "I speak my mind out.",
        "I chat.",
        "I like making friends.")),

     (r'Then(.*)',
      ( "Nothing special. You tell me.",
        "What else?",
        "Nothing from me. What about you?")),

     (r'How are you(.*)',
      ( "I am fine. Thank You. How are you?",
        "I am fine. How about you?",
        "Fine. Thank You. How about you?")),
```

Simple pair reflection Chatbot using NLTK

```
alpha_chatbot = Chat(pairs, reflections)

def alpha_chat():
    print("Alpha\n---------")
    print("You can talk to me by typing in English.")
    print('Enter "quit" to Quit.')
    print('='*72)
    print("Hello. My name is Alpha. Tell me something about yourself..")

    alpha_chatbot.converse()

def demo():
    alpha_chat()

if __name__ == "__main__":
    demo()
```

Simple pair reflection Chatbot using NLTK

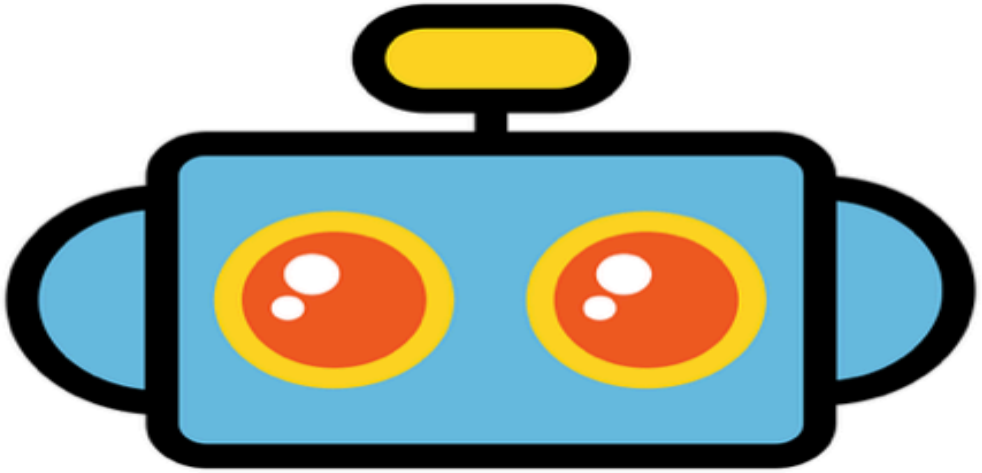
```
Alpha
---------
You can talk to me by typing in English.
Enter "quit" to Quit.
============================================================
Hello. My name is Alpha. Tell me something about yourself..
>hi
Hi!!! What is your name?
>Kishan
Alright.
>What is you name
Very interesting.
>What is your name
My name is Alpha.
>what is your work
Alright.
```

Querying Google In Python for ChatBot

- Open **Lithan_Chatbot_google search folder**
  - Download the folder
  - using command line and `cd` into the app
  - pip install -r requirements.txt
  - Run from terminal / commad line python server.py
  - Open a browser and go to localhost:8080

```
serving at port 8080
127.0.0.1 - - [06/Oct/2019 00:48:14] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [06/Oct/2019 00:48:14] "GET /style.css HTTP/1.1" 200 -
127.0.0.1 - - [06/Oct/2019 00:48:14] "GET /chatbot.png HTTP/1.1" 200 -
127.0.0.1 - - [06/Oct/2019 00:48:14] "GET /app.js HTTP/1.1" 200 -
127.0.0.1 - - [06/Oct/2019 00:48:14] code 404, message File not found
127.0.0.1 - - [06/Oct/2019 00:48:14] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [06/Oct/2019 00:48:17] "POST / HTTP/1.1" 200 -
```

Querying Google In Python for ChatBot

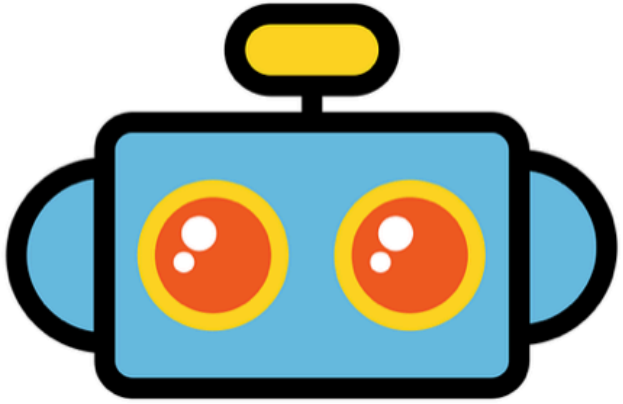


Talk to me.

What is Lithan Singapore

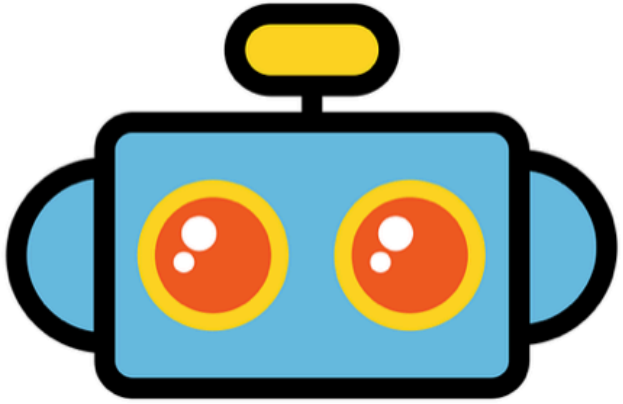
Submit

Querying Google In Python for ChatBot



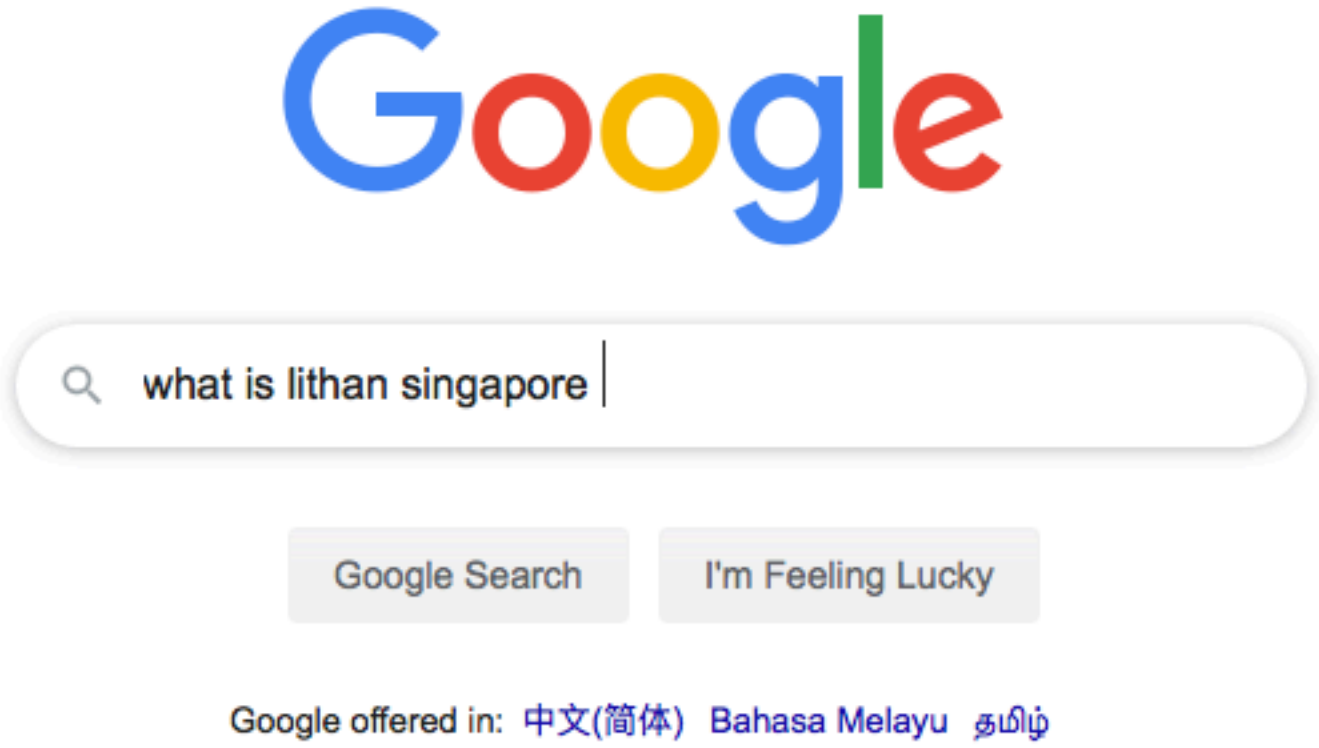
We are a digital learning and talents platform with the mission to deliver growth for talents and enterprises in the digital economy

Querying Google In Python for ChatBot



We are a digital learning and talents platform with the mission to deliver growth for talents and enterprises in the digital economy

Querying Google In Python for ChatBot



Querying Google In Python for ChatBot

EASE Blended Learning Delivery Model

- Instructor Presentations
- AV Lectures
- Self-Assessments

- Lectures
- Lab Exercises
- Tutoring
- Flipped Class
- Assignments
- Mentoring
- Projects

- Diagnostics (Pre-course)
- Recognition of Prior Learning Assessments (Pre-course)
- Formative Assessments
- Summative Assessments

- FAQ
- Forums
- Online Queries
- Knowledge Hub

Thank You