A

REPORT ON

"SKILL DEVELOPMENT LAB ON PYTHON & ARTIFICIAL INTELLIGENCE"

Submitted in partial fulfilment for the award of Degree of

Bachelor of Technology of Rajasthan Technical University, Kota

JAIPUR ENGINEERING COLLEGE
AND RESEARCH CENTRE

2021-22

Submitted to:

Mr. Ashish Sharma
(Assistant Professor)

Submitted by:

Satvik Jain
18EJCEC136

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING
JAIPUR ENGINEERING COLLEGE AND RESEARCH CENTRE, JAIPUR

ACKNOWLEDGEMENT

The success and final outcome of this lab required a lot of guidance and assistance from many people and we are extremely privileged to have gotten this all along the completion of our course. All that we have done is only due to such supervision and assistance and we would not forget to thank them. Deepest thanks to our Lab Supervisor **Mr. Ashish Sharma** for providing encouragement, constant support and guidance which was of a great help to complete this course successfully.

We also like to thank to all supporting ECE faculty member who has been a constant source of encouragement for successful completion of the project. Our warm thanks to Jaipur Engineering College and Research Centre for giving us this opportunity to work on our skills and knowledge through this project, also providing us with infrastructure and resources to improve ourself in various technical fields.

Satvik Jain

(18EJCEC136)

ECE, 4th Year, 8th Semester

TABLE OF CONTENTS

CHAPTER-1

INTRODUCTION TO PYTHON

1.1 Introduction

Python is a widely used high-level, general-purpose, interpreted, dynamic programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java. The language provides constructs intended to enable clear programs on both a small and large scale.

Python supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library. Python interpreters are available for installation on many operating systems, allowing Python code execution on a wide variety of systems.

Scripting Language:

A scripting or script language is a programming language that supports scripts, programs written for a special run-time environment that automate the execution of tasks that could alternatively be executed one-by-one by a human operator.

Scripting languages are often interpreted (rather than compiled). Primitives are usually the elementary tasks or API calls, and the language allows them to be combined into more complex programs. Environments that can be automated through scripting include software applications, web pages within a web browser, the shells of operating systems (OS), embedded systems, as well as numerous games.

A scripting language can be viewed as a domain-specific language for a particular environment; in the case of scripting an application, this is also known as an **extension language**. Scripting languages are also sometimes referred to as very high-level programming languages, as they operate at a high level of abstraction, or as control languages.

Object Oriented Programming Language:

Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which may contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods. A distinguishing feature of objects is that an object's procedures can access and often modify the data fields of the object with which they are associated (objects have a notion of "this" or "self").

In OO programming, computer programs are designed by making them out of objects that interact with one another. There is significant diversity in object- oriented programming, but most popular languages are class-based, meaning that objects are instances of classes, which typically also determines their type.

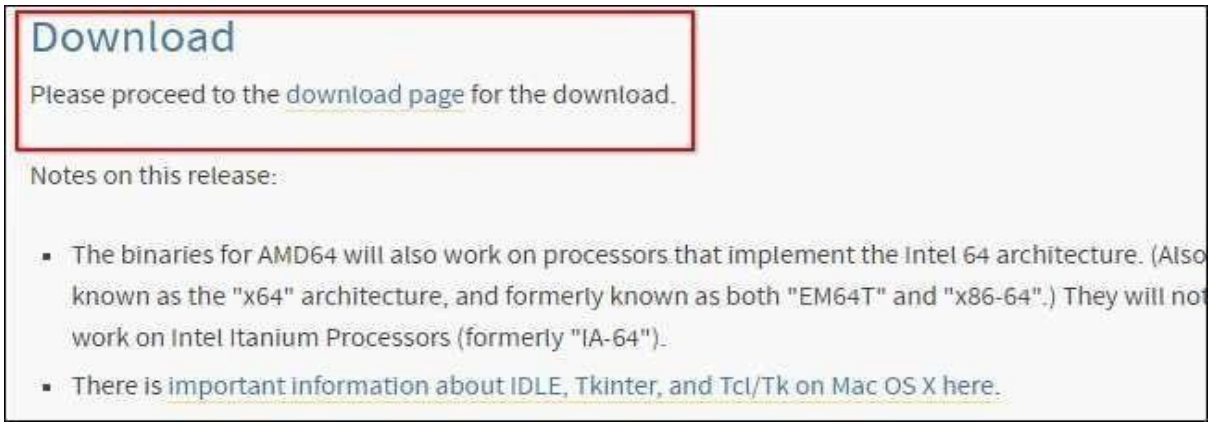
Python Releases:

Python	1.0	- January 1994
Python	1.5	- December 31, 1997
Python	1.6	- September 5, 2000
Python	2.0	- October 16, 2000
Python	2.1	- April 17, 2001
Python	2.2	- December 21, 2001
Python	2.3	- July 29, 2003
Python	2.4	- November 30, 2004
Python	2.5	- September 19, 2006
Python	2.6	- October 1, 2008
Python	2.7	- July 3, 2010
Python	3.0	- December 3, 2008
Python	3.1	- June 27, 2009
Python	3.2	- February 20, 2011
Python	3.3	- September 29, 2012
Python	3.4	- March 16, 2014
Python	3.5	- September 13, 2015
Python	3.6	- December 23, 2016

1.2 Downloading and Installing Python

If you don't already have a copy of Python installed on your computer, you will need to open up your Internet browser and go to the Python download page ***(http://www.python.org/download/).***

Once you have clicked on that, you will be taken to a page with a description of all the new updates and features of 3.4.1, however, you can always read that while the download is in process. Scroll to the bottom of the page till you find the "Download" section and click on the link that says "download page."

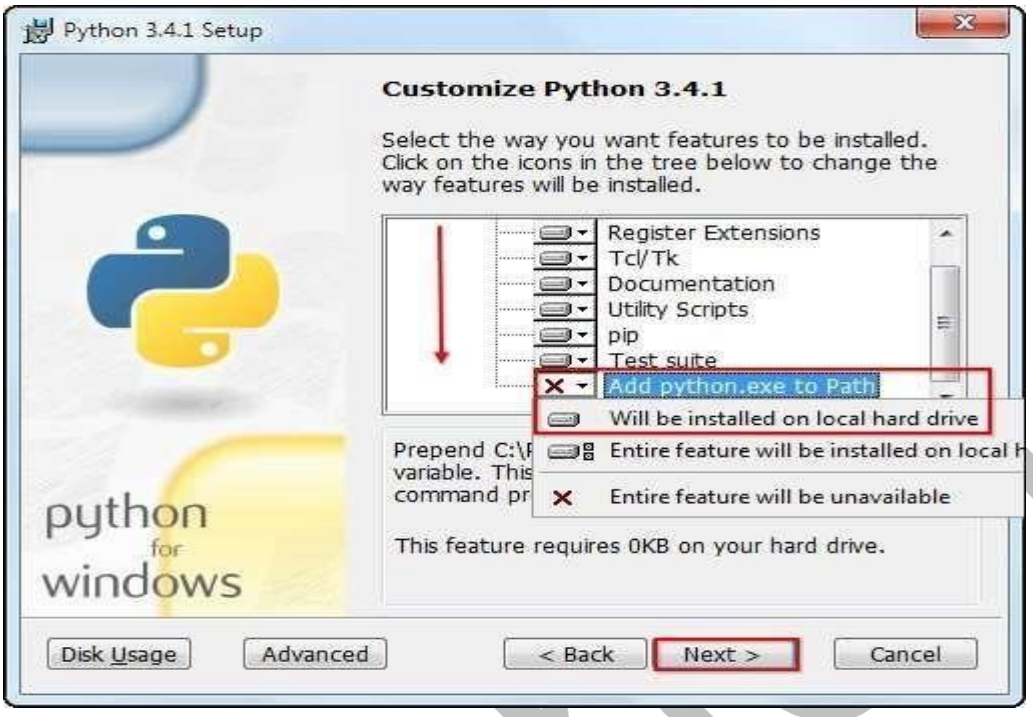


Now you will scroll all the way to the bottom of the page and find "Windows x86 MSI installer." If you want to download the 86-64 bit MSI, feel the free to do so. We believe that even if you have a 64-bit operating system installed on your computer, the 86-bit MSI is preferable. We say this because it will still run well and sometimes, with the 64-bit architectures, some of the compiled binaries and Python libraries don't work well.

Once you have downloaded the Python MSI, simply navigate to the download location on your computer, double clicking the file and pressing Run when the dialog box pops up.


Open File - Security Warning
Do you want to run this file?
Name: C:\Users\Jenny\Downloads\python-3.4.1.msi
Publisher: Python Software Foundation
Type: Windows Installer Package
From: C:\Users\Jenny\Downloads\python-3.4.1.msi
Run
Cancel
Always ask before opening this file
While files from the Internet can be useful, this file type can potentially harm your computer. Only run software from publishers you trust. What's the risk?


Python 3.4.1 Setup
Select whether to install Python 3.4.1 for all users of this computer.
Install for all users
Install just for me (not available on Windows Vista)
python
for
windows
Back
Next >
Cancel



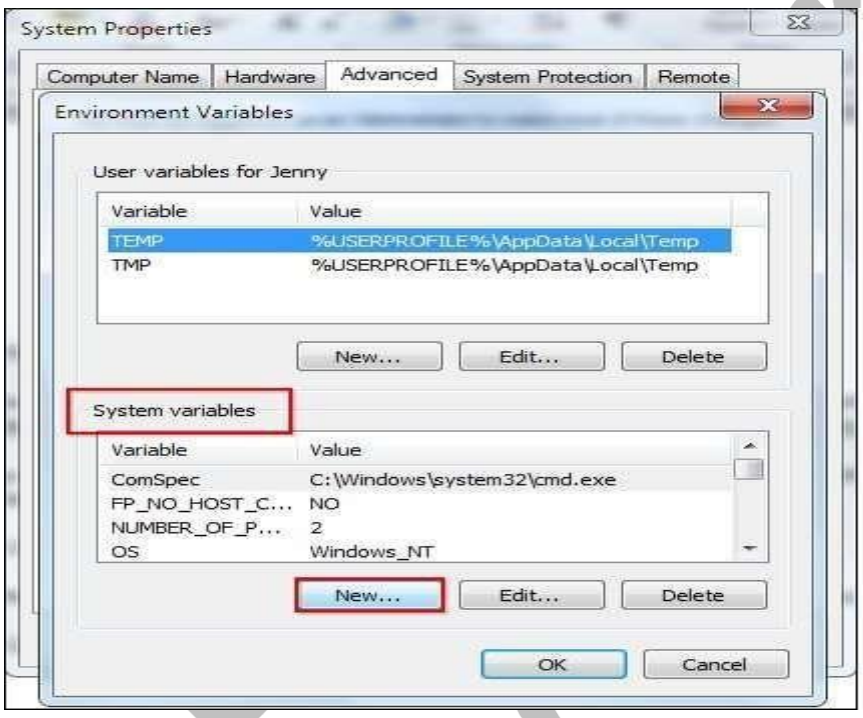
Scroll down in the window and find the “Add Python.exe to Path” and click on the small red “x.” Choose the “Will be installed on local hard drive” option then press “Next.”

Setup the Path Variable-

Begin by opening the start menu and typing in “environment” and select the option called “Edit the system environment variables.”

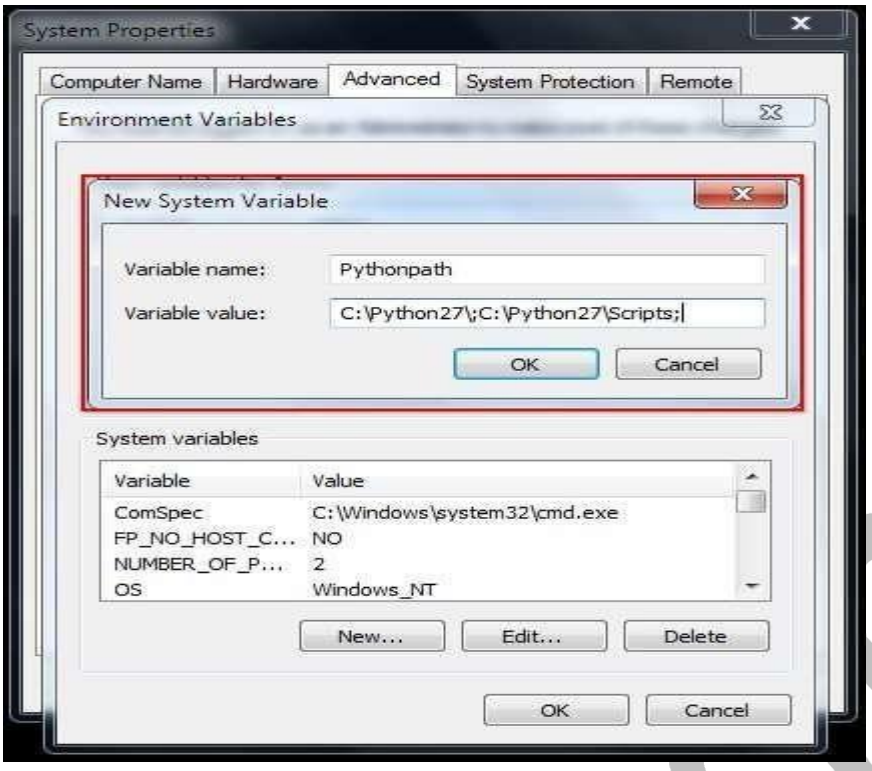
When the “System Properties” window appears, click on “Environment Variables…”

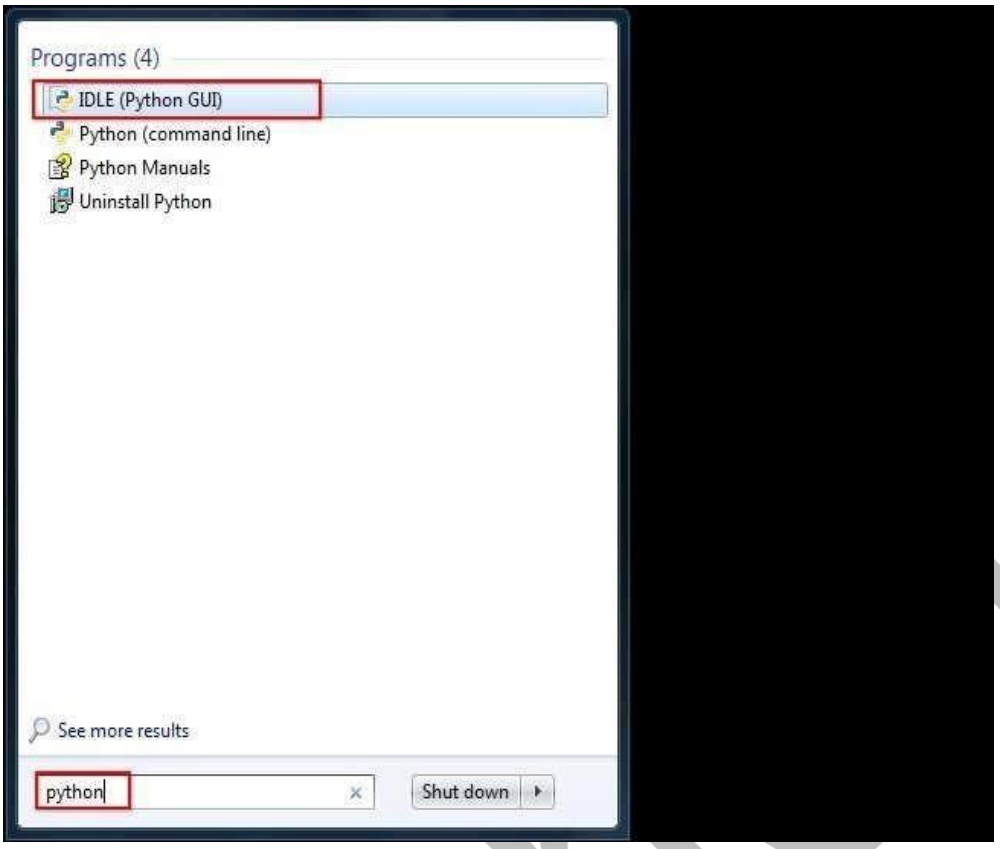
Once you have the “Environment Variables” window open, direct your focus to the bottom half. You will notice that it controls all the “System Variables” rather than just this associated with your user. Click on “New…” to create a new variable for Python.

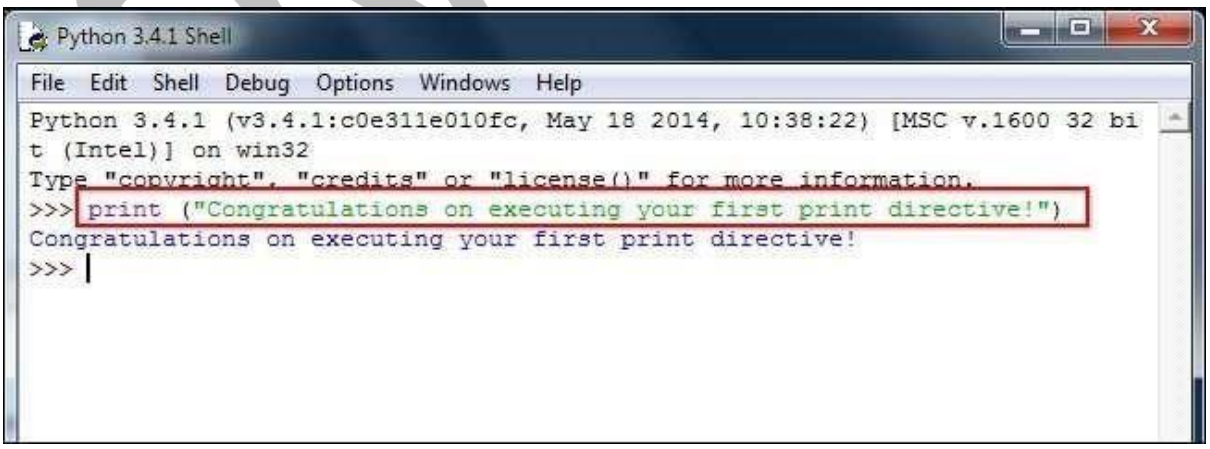


Simply enter a name for your Path and the code shown below. For the purposes of this example we have installed Python 2.7.3, so we will call the path: “Pythonpath.”

The string that you will need to enter is: “C:\Python27\;C:\Python27\Scripts;”

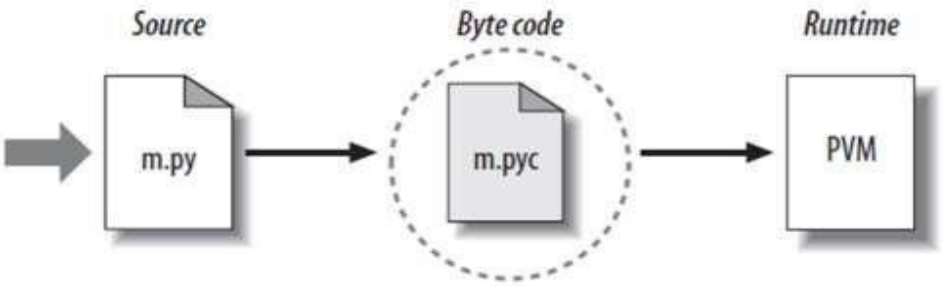

System Properties
Computer Name
Hardware
Advanced
System Protection
Remote
Environment Variables
New System Variable
Variable name:
Pythonpath
Variable value:
C:\Python27\;C:\Python27\Scripts;
OK
Cancel
System variables
Variable
Value
ComSpec
C:\Windows\system32\cmd.exe
FP_NO_HOST_C...
NO
NUMBER_OF_P...
2
OS
Windows_NT
New...
Edit...
Delete
OK
Cancel





Running The Python ID

Now that we have successfully completed the installation process and added our“Environment Variable,” you are ready to create your first basic Python script. Let’s begin by opening Python’s GUI by pressing “Start” and typing “Python” and selecting the “IDLE (Python GUI).”



Python Code Execution

Python’s traditional runtime execution model: source code you type is translated to byte code, which is then run by the Python Virtual Machine. Your code is automatically compiled, but then it is interpreted.

1.3 Data Type

Data types determine whether an object can do something, or whether it just would not make sense. Other programming languages often determine whether an operation makes sense for an object by making sure the object can never be stored somewhere where the operation will be performed on the object (this type system is called static typing). Python does not do that.

Instead it stores the type of an object with the object, and checks when the operation is performed whether that operation makes sense for that object (this is called dynamic typing).

Python has many native data types. Here are the important ones:

Booleans are either True or False.

Numbers can be integers (1 and 2), floats (1.1 and 1.2), fractions (1/2 and 2/3),

or even complex numbers.

Strings are sequences of Unicode characters, e.g. an HTML document,

Bytes and byte arrays, e.g. a JPEG image file.

Lists are ordered sequences of values.

Tuples are ordered, immutable sequences of values.

Sets are unordered bags of values.

Variables

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

Ex:

counter = 100 # An integer assignment

miles = 1000.0 # A

floating point

```
name    = "John"                    # A string
```

String

In programming terms, we usually call text a string. When you think of a string as a collection of letters, the term makes sense.

All the letters, numbers, and symbols in this book could be a string. For that matter, your name could be a string, and so could your address.

Creating Strings

In Python, we create a string by putting quotes around text. For example, we could take our otherwise useless.

•	"hello"+"world"	"helloworld"	# concatenation
•	"hello"*3	"hellohellohello"	# repetition
•	"hello"[0]	"h"	# indexing
•	"hello"[1:4]	"ell"	# slicing
•	len("hello")	5	# size
•	"hello" < "jello"	1	# comparison
•	"e" in "hello"	1	# search

Operator	Meaning	Example
+	Add two operands or unary plus	x + y +2
-	Subtract right operand from the left or unary minus	x - y - 2
*	Multiply two operands	x * y
/	Divide left operand by the right one (always results into float)	x / y
%	Modulus - remainder of the division of left operand by the right	x % y (remainder of x/y)
//	Floor division - division that results into whole number adjusted to the left in the number line	x // y
**	Exponent - left operand raised to the power of right	x**y (x to the power y)

Comparison Operator

>	Greater that - True if left operand is greater than the right	x > y
<	Less that - True if left operand is less than the right	x < y
==	Equal to - True if both operands are equal	x == y
!=	Not equal to - True if operands are not equal	x != y

>=	Greater than or equal to - True if left operand is greater than or equal to the right	x >= y
<=	Less than or equal to - True if left operand is less than or equal to the right	x <= y

Logical Operator

Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

1.4 Tuples

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses also. For example −

```
tup1 = ('physics', 'chemistry', 1997, 2000);
```

```
tup2 = (1, 2, 3, 4, 5 ); tup3 = "a",
"b", "c", "d"; Accessing Values
in Tuples:
```

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example −

```
tup1 = ('physics', 'chemistry', 1997, 2000);
```

tup2 = (1, 2, 3, 4, 5, 6, 7); print

"tup1[0]: ", tup1[0] print

"tup2[1:5]: ", tup2[1:5]

When the above code is executed, it produces the following result −

tup1[0]: physics tup2[1:5]:

[2, 3, 4, 5]

Basic Tuples Operations

Tuples respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new tuple, not a string. In fact, tuples respond to all of the general sequence operations we used on strings in the prior chapter −

Python Expression	**Results**	**Description**
len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1, 2, 3): print x,	1 2 3	Iteration

Built-in Tuple Functions

Python includes the following tuple functions −

SN	Function with Description
1	**cmp(tuple1, tuple2)** Compares elements of both tuples.
2	**len(tuple)** Gives the total length of the tuple.
3	**max(tuple)** Returns item from the tuple with max value.
4	**min(tuple)** Returns item from the tuple with min value.
5	**tuple(seq)** Converts a list into tuple.

List

The list is a most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.

Creating a list is as simple as putting different comma-separated values

between square brackets. For example − list1 = ['physics', 'chemistry',

1997, 2000]; list2 = [1, 2,

3, 4, 5]; list3 = ["a", "b",

"c", "d"];

Similar to string indices, list indices start at 0, and lists can be sliced, concatenated and so on.

Accessing Values in Lists:

To access values in lists, use the square brackets for slicing along with the index or

indices to obtain value available at that index. For example −

list1 = ['physics', 'chemistry', 1997, 2000];

list2 = [1, 2, 3, 4, 5, 6, 7]; print "list1[0]:
", list1[0] print
"list2[1:5]: ", list2[1:5]
Output:list1[0]: physics
list2[1:5]: [2, 3, 4, 5]

Update: list = ['physics', 'chemistry', 1997, 2000];

print "Value available at index 2 : " print list[2]
list[2] = 2001; print "New value available at index 2
: " print list[2]

Output: Value available at index 2 : 1997

New value available at index 2 : 2001

Delete: list1 = ['physics', 'chemistry', 1997, 2000];

print
list1
del
list1[2];
print
"After
deleting
value at
index 2 : "
print list1

['physics', 'chemistry', 1997, 2000] **Output:**After deleting value at index 2 :

['physics', 'chemistry', 2000]

Basic List Operation

Python Expression	Results	Description
len([1, 2, 3])	3	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Membership
for x in [1, 2, 3]: print x,	1 2 3	Iteration

Built-in List Functions & Methods:

SN	Function with Description
1	**cmp(list1, list2)** Compares elements of both lists.
2	**len(list)** Gives the total length of the list.
3	**max(list)** Returns item from the list with max value.
4	**min(list)** Returns item from the list with min value.
5	**list(seq)** Converts a tuple into list.

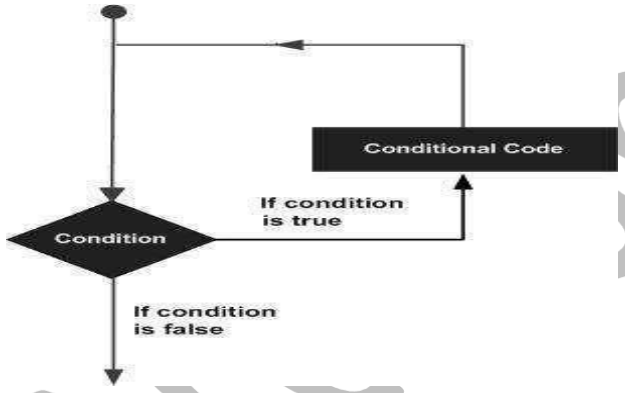
Python includes following list methods

SN	Methods with Description
1	**list.append(obj)** Appends object obj to list
2	**list.count(obj)** Returns count of how many times obj occurs in list
3	**list.extend(seq)** Appends the contents of seq to list
4	**list.index(obj)** Returns the lowest index in list that obj appears
5	**list.insert(index, obj)** Inserts object obj into list at offset index
6	**list.pop(obj=list[-1])** Removes and returns last object or obj from list
7	**list.remove(obj)** Removes object obj from list
8	**list.reverse()** Reverses objects of list in place
9	**list.sort([func])** Sorts objects of list, use compare func if given

1.5 Loops in Python

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times. The following diagram illustrates a loop statement −



Python programming language provides following types of loops to handle looping requirements.

nested loops	You can use one or more loop inside any anotherwhile, for or do..while loop.

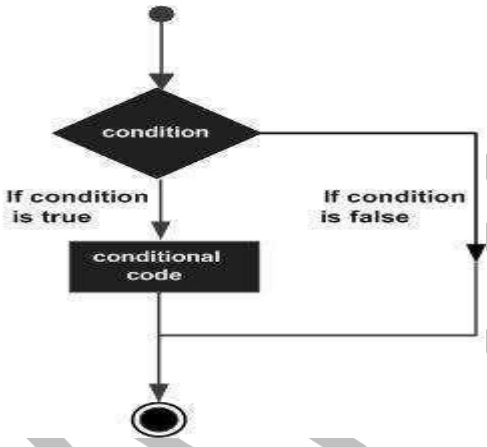
Loop Type	Description
while loop	Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.

for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

Conditional Statements

Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions.Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome. You need to determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise.

Python programming language provides following types of decision making statements.



Statement	Description
if statements	An **if statement** consists of a boolean expression followed by one or more statements.

if...else statements	An **if statement** can be followed by an optional **else statement**, which executes when the boolean expression is FALSE.
nested if statements	You can use one **if** or **else if** statement inside another **if** or **else if** statement(s).

1.6 Scope of Python

Science- Bioinformatics

System Administration

-Unix

-Web logic

-Web sphere

Web Application Development

-CGI Testing scripts

What Can We do With Python?

System programming

Graphical User Interface Programming

Internet Scripting

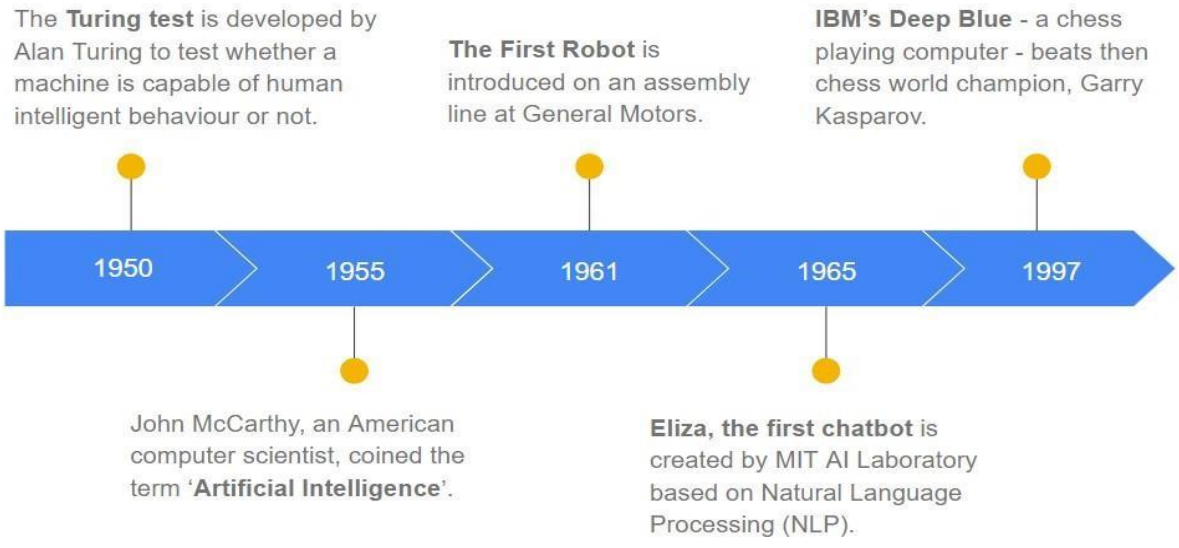
Component

CHAPTER-2

ARTIFICIAL INTELLIGENCE

2.1 History of AI

The term artificial intelligence was first coined by John McCarthy in 1956 when he held the first academic conference on the subject. But the journey to understand if machines can truly think began much before that. In Vannevar Bush's seminal work As We May Think [Bush45] he proposed a system which amplifies people's own knowledge and understanding. Five years later Alan Turing wrote a paper on the notion of machines being able to simulate human beings and the ability to do intelligent things, such as play Chess [Turing50]. No one can refute a computer's ability to process logic. But to many it is unknown if a machine can think. The precise definition of think is important because there has been some strong opposition as to whether or not this notion is even possible. For example, there is the socalled 'Chinese room' argument [Searle80]. Imagine someone is locked in a room, where they were passed notes in Chinese. Using an entire library of rules and look-up tables they would be able to produce valid responses in Chinese, but would they really 'understand' the language? The argument is that since computers would always be applying rote fact lookup they could never 'understand' a subject. This argument has been refuted in numerous ways by researchers, but it does undermine people's faith in machines and so-called expert systems in lifecritical applications.



2.2 Themes of AI

The main advances over the past sixty years have been advances in search algorithms, machine learning algorithms, and integrating statistical analysis into understanding the world at large.

However most of the breakthroughs in AI aren't noticeable to most people. Rather than talking machines used to pilot space ships to Jupiter, AI is used in more subtle ways such as examining purchase histories and influence marketing decisions [Shaw01]. What most people think of as 'true AI' hasn't experienced rapid progress over the decades. A common theme in the field has been to overestimate the difficulty of foundational problems. Significant AI breakthroughs have been promised 'in 10 years' for the past 60 years. In addition, there is a tendency to redefine what 'intelligent' means after machines have mastered an area or problem. This so-called 'AI Effect' contributed to the downfall of USbased AI research in the 80s. In the field of AI expectations seem to always outpace the reality. After decades of research, no computer has come close to passing the Turing Test (a model for measuring 'intelligence'); Expert Systems have grown but have not become as common as human experts; and while we've built software that can beat humans at some games, open ended games are still far from the mastery of computers. Is the problem simply that we haven't focused enough resources on basic research, as is seen in the AI winter section, or is the complexity of AI one that we haven't yet come to grasp yet? (And instead, like in the case of computer Chess, we focus on much more specialized problems rather than understanding the notion of 'understanding' in a problem domain.) This paper will go into some of these themes to provide a better understanding for the field of AI and how it has developed over the years. In looking at some of the key areas of AI work and the forces that drove them, perhaps we can better understand future developments in the field.

2.3 The Turing Test

Introduction

The Turing test is a central, long term goal for AI research – will we ever be able to build a computer that can sufficiently imitate a human to the point where a suspicious judge cannot tell the difference between human and machine? From its inception it has followed a path similar to much of the AI research. Initially it looked to be difficult but possible (once

hardware technology reached a certain point), only to reveal itself to be far more complicated than initially thought with progress slowing to the point that some wonder if it will ever be reached. Despite decades of research and great technological advances the Turing test still sets a goal that AI researchers strive toward while finding along the way how much further we are from realizing it. In 1950 English Mathematician Alan Turing published a paper entitled “Computing Machinery and Intelligence” which opened the doors to the field that would be called AI. This was years before the community adopted the term Artificial Intelligence as coined by John McCarthy [2]. The paper itself began by posing the simple question, “Can machines think?” *1+. Turing then went on to propose a method for evaluating whether machines can think, which came to be known as the Turing test. The test, or “Imitation Game” as it was called in the paper, was put forth as a simple test that could be used to prove that machines could think. The Turing test takes a simple pragmatic approach, assuming that a computer that is indistinguishable from an intelligent human actually has shown that machines can think. The idea of such a long term, difficult problem was a key to defining the field of AI because it cuts to the heart of the matter – rather than solving a small problem it defines an end goal that can pull research down many paths. Without a vision of what AI could achieve, the field itself might never have formed or simply remained a branch of math or philosophy. The fact that the Turing test is still discussed and researchers attempt to produce software capable of passing it are indications that Alan Turing and the proposed test provided a strong and useful vision to the field of AI. It’s relevance to this day seems to indicate that it will be a goal for the field for many years to come and a necessary marker in tracking the progress of the AI field as a whole. This section will explore the history of the Turing test, evaluate its validity, describe the current attempts at passing it and conclude with the possible future directions the Turing test solution may take.

ARTIFICIAL INTELLIGENCE

Artificial Intelligence is the ability to design smart machines or to develop self-learning software applications that imitate the traits of the human mind like reasoning, problem-solving, planning, optimal decision making, sensory perceptions etc. The capacity of artificial intelligent approaches to outperform human actions in terms of knowledge discovery gained the attention of business and research communities all over the world, and this field of study witnessed rapid progress in the past two decades. let us move ahead in this introduction to AI post in detail.

While a number of definitions of artificial intelligence (AI) have surfaced over the last few decades, John McCarthy offers the following definition in this 2004 paper (PDF, 106 KB) (link resides outside IBM), " It is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable."

However, decades before this definition, the birth of the artificial intelligence conversation was denoted by Alan Turing's seminal work, "Computing Machinery and Intelligence" (PDF, 89.8 KB) (link resides outside of IBM), which was published in 1950. In this paper, Turing, often referred to as the "father of computer science", asks the following question, "Can machines think?" From there, he offers a test, now famously known as the "Turing Test", where a human interrogator would try to distinguish between a computer and human text response. While this test has undergone much scrutiny since its publish, it remains an important part of the history of AI as well as an ongoing concept within philosophy as it utilizes ideas around linguistics.

Types Of Learning In Artificial Intelligence

1. Artificial Narrow Intelligence
2. Artificial General Intelligence
3. Artificial Super Intelligence

These are the three stages through which AI can evolve, rather than the 3 types of Artificial Intelligence.

Artificial Narrow Intelligence (ANI)

Also known as Weak AI, ANI is the stage of Artificial Intelligence involving machines that can perform only a narrowly defined set of specific tasks. At this stage, the machine does not possess any thinking ability, it just performs a set of pre-defined functions Examples of Weak AI include Siri, Alexa, Self-driving cars, Alpha-Go, Sophia the humanoid and so on.
Almost all the AI-based systems built till this date fall under the category of Weak AI.

Artificial General Intelligence (AGI)

AGI has the capability to train, learn, understand and perform functions just like a normal human does. These systems will have multi-functional capabilities cutting across different domains. These systems will be more agile and will react and improvise just like humans while facing unprecedented scenarios. There is no real-world example of this kind of AI, but a good amount of progress has been made in this field.

Artificial Super Intelligence (ASI)

Artificial Super Intelligence will be the top-most point of AI development. ASI will be the most potent form of intelligence to ever exist on this planet. It will be able to perform all the tasks better than humans because of its inordinately superior data processing, memory, and decision-making ability. Some of the researchers fear that the advent of ASI will ultimately result in "Technological Singularity". It is a hypothetical situation in which the growth in technology will reach an uncontrollable stage, resulting in an unimaginable change in Human Civilization.

2.4 Types Of AI

Based on the functionality of AI-based systems, AI can be categorized into the following types:

1. Reactive Machines AI
2. Limited Memory AI
3. Theory of Mind AI
4. Self-aware AI

1. Reactive Machine

They are the most basic and oldest type of Artificial Intelligence. They replicate a human's ability to react to different kinds of stimuli. This type of AI has no memory power, so they lack the capability to use previously gained information/experience to obtain better results.

Therefore, these kinds of AI don't have the ability to train themselves like the ones we come across nowadays.

Example: Deep Blue, IBM's chess-playing supercomputer

2. Limited Theory

This type of AI, along with the ability of Reactive Machines, have memory capabilities so they can use past information/experience to make better future decisions. Most of the common applications existing around us fall under this category. These AI applications can be trained by a large volume of training data they store in their memory in a reference model.

Example: Limited Memory technology is used in many self-driving cars use

3. Theory of Mind

Theory of Mind is the next level of AI, which has very limited to no presence in our day-today lives. These kind of AI are mostly in the "Work in Progress" stage and are usually confined to research labs. These kinds of AI, once developed, will have a very deep understating of human minds ranging from their needs, likes, emotions, thought process, etc. Basis their understanding of Human minds and their whims, the AI will be able to alter its own response.

Example: Researcher Winston in his research, showed a prototype of a robot that can walk down the small corridor with other robots coming from the opposite direction; the AI can foresee other robots movements and can turn right, left or any other way so as to avoid a possible collision with the incoming robots. As per Wilson, this Robot determines its action based on its "common sense" of how other robots will move.

4. Self-Aware AI

This is the final stage of AI. Its current existence is only hypothetical and can be found only in Science fiction movies. These kinds of AI can understand and evoke human emotions and have emotions of their own. These kind of AI are decades, if not centuries, away from materializing. It is this kind of AI which AI skeptics like Elon Musk are wary of. This is because once it is selfaware, the AI can get into Self-Preservation mode; it might consider humanity as a potential threat and may directly or indirectly pursue endeavor to end humanity.

Weak AI vs. Strong AI FFF

Weak AI—also called Narrow AI or Artificial Narrow Intelligence (ANI)—is AI trained and focused to perform specific tasks. Weak AI drives most of the AI that surrounds us today. 'Narrow' might be a more accurate descriptor for this type of AI as it is anything but weak; it enables some very robust applications, such as Apple's Siri, Amazon's Alexa, IBM Watson, and autonomous vehicles.

Strong AI is made up of Artificial General Intelligence (AGI) and Artificial Super Intelligence (ASI). Artificial general intelligence (AGI), or general AI, is a theoretical form of AI where a machine would have an intelligence equalled to humans; it would have a self-aware consciousness that has the ability to solve problems, learn, and plan for the future. Artificial Super Intelligence (ASI)—also known as superintelligence—would surpass the intelligence and ability of the human brain. While strong AI is still entirely theoretical with no practical examples in use today, that doesn't mean AI researchers aren't also exploring its development.

2.5 Artificial Intelligence vs Machine Learning vs Deep Learning

AI:

AI is a broader concept of machines being able to behave like humans would. Human are able to intelligently analyze a lot of variables around them and act accordingly. But, human mind is not just about analysis. It is also about learning based on that analysis and getting better at it. It is about using wisdom to make judgments and act accordingly. AI is about that science. It is that concept.

AI has been around for a while. When we first built computers, we were able to replicate some aspects of human brain based on the understanding at that time. However, as we learnt more and more about our brains, the concept of AI has changed. We have moved on from just replicating increasingly complex calculations to mimicking a human decision making process and intelligent actions based on that. So now, AI is not about just identifying the best driving route based on traffic conditions - it is about building a system that will drive the car like a human would and take decisions along the way, especially in an emergency situation.

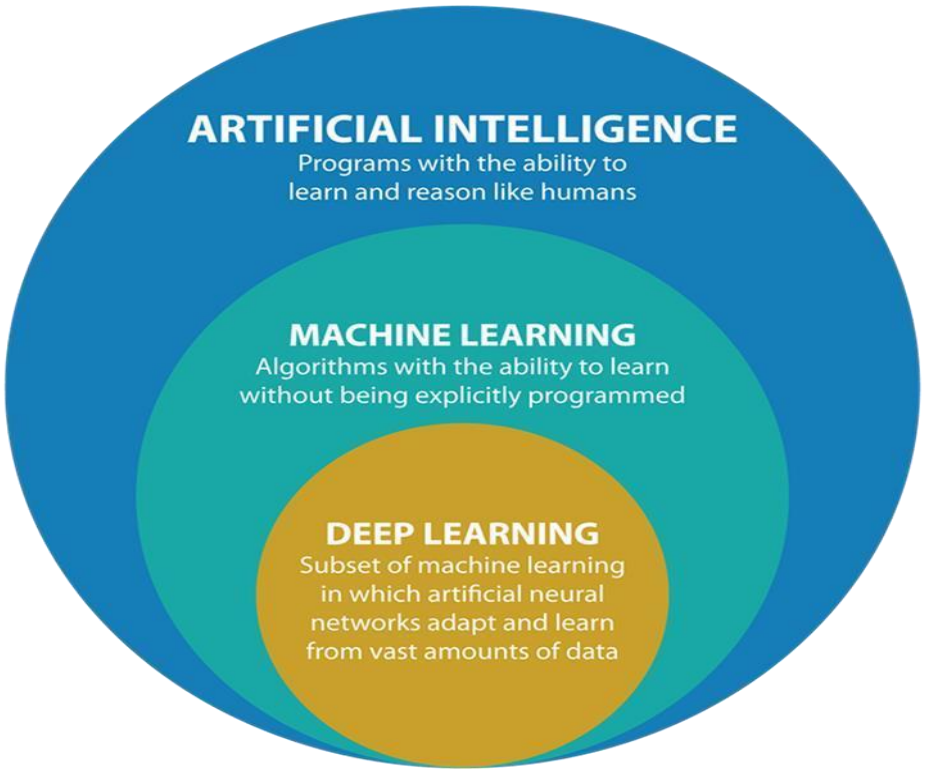
ML:

ML is a subset of AI. It refers to a system that is able to learn by itself without any human intervention. As you can already see, every AI based solution may need this. However, it is not mandatory (though highly desirable). The more a system learns, the better it is at providing an output.

It is used at many places. When Netflix suggests some movies for you to watch, it is doing so based on what it has learnt from the user experiences and ratings of all of Netflix's users and your personal movie watching history. It can do so based on an insane amount of data available to it and using specific ML algorithms to come up with suggestions. These algorithms are first created by humans and fine tuned. But once they put into production, they can be auto-fine-tuned based on the newer data that keeps getting generated every minute.

DL:

DL is a subset of ML. It refers to ML applied to a very large data set (usually labelled data set). The concepts used in DL are same as ML and everything associated with ML applies to DL as well - the difference being the amount of data being processed. Because of this, DL requires huge amount of processing power.



2.6 Artificial Intelligence Applications

There are numerous, real-world applications of AI systems today. Below are some of the most common examples:

- **Speech recognition:** It is also known as automatic speech recognition (ASR), computer speech recognition, or speech-totext, and it is a capability which uses natural language processing (NLP) to process human speech into a written format. Many mobile devices incorporate speech recognition into their systems to conduct voice search—e.g. Siri—or provide more accessibility around texting.

- **Customer service:** Online virtual agents are replacing human agents along the customer journey. They answer frequently asked questions (FAQs) around topics, like shipping, or provide personalized advice, cross-selling products or suggesting sizes for users, changing the way we think about customer engagement across websites and social media platforms. Examples include messaging bots on ecommerce sites with virtual agents, messaging apps, such as Slack and Facebook Messenger, and tasks usually done by virtual assistants and voice assistants.

- **Computer vision:** This AI technology enables computers and systems to derive meaningful information from digital images, videos and other visual inputs, and based on those inputs, it can take action. This ability to provide recommendations distinguishes it from image recognition tasks. Powered by convolutional neural networks, computer vision has applications within photo tagging in social media, radiology imaging in healthcare, and self-driving cars within the automotive industry.

- **Recommendation engines:** Using past consumption behavior data, AI algorithms can help to discover data trends that can be used to develop more effective crossselling strategies. This is used to make relevant add-on recommendations to customers during the checkout process for online retailers.

- **Automated stock trading:** Designed to optimize stock portfolios, AI-driven highfrequency trading platforms make thousands or even millions of trades per day without human intervention.

CHAPTER-3

PROJECT IMPLEMENTED (AI Chat Bot with NPL)

3.1 Introduction

Were you ever curious as to how to build a talking ChatBot with Python and also have a conversation with your own personal AI?

As the topic suggests we are here to help you have a conversation with your AI today. To have a conversation with your AI, you need a few pre-trained tools which can help you build an AI chatbot system. In this article, we will guide you to combine speech recognition processes with an artificial intelligence algorithm.

Natural Language Processing or NLP is a prerequisite for our project. NLP allows computers and algorithms to understand human interactions via various languages. In order to process a large amount of natural language data, an AI will definitely need NLP or Natural Language Processing. Currently, we have a number of NLP research ongoing in order to improve the AI chatbots and help them understand the complicated nuances and undertones of human conversations.

Chatbots are nothing but applications that are used by businesses or other entities to conduct an automatic conversation between a human and an AI. These conversations may be via text or speech. Chatbots are required to understand and mimic human conversation while interacting with humans from all over the world. From the first chatbot to be created ELIZA to Amazon's ALEXA today, chatbots have come a long way. In this tutorial, we are going to cover all the basics you need to follow along and create a basic chatbot that can understand human interaction and also respond accordingly. We will be using speech recognition APIs and also pre-trained Transformer models.

3.2 Whats NPL?

NLP stands for Natural Language Processing. Using NLP technology, you can help a machine understand human speech and spoken words. NLP combines computational linguistics that is the rule-based modelling of the human spoken language with intelligent algorithms such as statistical, machine, and deep learning algorithms. These technologies together create the smart voice assistants and chatbots that you may be used in everyday life.

There are a number of human errors, differences, and special intonations that humans use every day in their speech. NLP technology allows the machine to understand, process, and respond to large volumes of text rapidly in real-time. In everyday life, you have encountered NLP tech in voice-guided GPS apps, virtual assistants, speech-to-text note creation apps, and other app support

chatbots. This tech has found immense use cases in the business sphere where it's used to streamline processes, monitor employee productivity, and increase sales and after-sales efficiency.

3.3 Talks in NPL

The task of interpreting and responding to human speech is filled with a lot of challenges that we have discussed in this article. In fact, it takes humans years to overcome these challenges and learn a new language from scratch. To overcome these challenges, programmers have integrated a lot of functions to the NLP tech to create useful technology that you can use to understand human speech, process, and return a suitable response.

NLP tasks are responsible for breaking down human text and audio signals from voice data in ways that can be analysed and converted into data that the computer understands. Some of the tasks included in NLP data ingestion are as follows:

- **Speech recognition:** speech recognition or speech to text conversion is an incredibly important process involved in speech analysis. Speech tagging or grammatical tagging is a subprocess of speech recognition that allows a computer to break down speech and tag it with implied context, accent or other speech definition points.
- **Word sense Disambiguation:** In human speech, a word may have multiple meanings. The process of word sense disambiguation is a semantic analysis that selects the meaning of a given word that best suits it in the given context. For example, this process assists in deciding whether a word is a verb or a pronoun.
- **Named Entity Recognition or NEM:** NEM identifies words and phrases as useful entities for example, 'Dev' is a person's name and 'America' is the name of a country.

- **Sentiment analysis:** Human speech often contains sentiments and undertones Extracting these undertones and hidden contexts such as attitude, sarcasm, fear or joy is perhaps the most difficult task that is undertaken by NLP processes.

3.4 Types of Chat bots

Chatbots are a relatively recent concept and despite having a huge number of programs and NLP tools, we basically have just two different categories of chatbots based on the NLP technology that they utilize. These two types of chatbots are as follows:

- **Scripted chatbots:** Scripted chatbots are classified as chatbots that work on pre-determined scripts that are created and stored in their library. Whenever a user types a query or speaks a query (in the case of chatbots equipped with speech to text conversion modules), the chatbot responds to this query according to the pre-determined script that is

stored within its library.One of the cons of such a chatbot is the fact that user needs to provide their query in a very structured manner with comma-separated commands or other forms of a regular expression that makes it easier for the bot to perform string analysis and understand the query. This makes this kind of chatbot difficult to integrate with NLP aided speech to text conversion modules. Hence, these chatbots can hardly ever be converted into smart virtual assistants.

- **Artificially Intelligent Chatbots:** Artificially intelligent chatbots, as the name suggests, are created to mimic human-like traits and responses. NLP or Natural Language Processing is hugely responsible for enabling such chatbots to understand the dialects and undertones of human conversation. NLP combined with artificial intelligence creates a truly intelligent chatbot that can respond to nuanced questions and learn from every interaction to create better-suited responses the next time.The AI chatbots have been developed to assist human users on different platforms such as automated chat support or virtual assistants helping with a song or restaurant selection.

-

3.5 Source Code

```
# for speech-to-text

import speech_recognition as sr


# for text-to-speech

from gtts import gTTS


# for language model

import transformers


import os

import time


# for data

import os

import datetime

import numpy as np



# Building the AI
```

```
class ChatBot():
    def __init__(self, name):
        print("----- Starting up", name, "-----")
        self.name = name

    def speech_to_text(self):
        recognizer = sr.Recognizer()
        with sr.Microphone() as mic:
            print("Listening...")
            audio = recognizer.listen(mic)
            self.text="ERROR"
        try:
            self.text = recognizer.recognize_google(audio)
            print("Me  --> ", self.text)
        except:
            print("Me  -->  ERROR")

    @staticmethod
    def text_to_speech(text):
        print("Dev --> ", text)
        speaker = gTTS(text=text, lang="en", slow=False)
```

```
        speaker.save("res.mp3")
        statbuf = os.stat("res.mp3")
        mbytes = statbuf.st_size / 1024
        duration = mbytes / 200
        os.system('start res.mp3')  #if you are using mac->afplay or else for windows->start
        # os.system("close res.mp3")
        time.sleep(int(50*duration))
        os.remove("res.mp3")


    def wake_up(self, text):
        return True if self.name in text.lower() else False

    @staticmethod
    def action_time():
        return datetime.datetime.now().time().strftime('%H:%M')


# Running the AI
if __name__ == "__main__":

    ai = ChatBot(name="dev")
```

```
nlp = transformers.pipeline("conversational", model="microsoft/DialoGPT-medium")
os.environ["TOKENIZERS_PARALLELISM"] = "true"

ex=True
while ex:
    ai.speech_to_text()

    ## wake up
    if ai.wake_up(ai.text) is True:
        res = "Hello I am Dave the AI, what can I do for you?"

    ## action time
    elif "time" in ai.text:
        res = ai.action_time()

    ## respond politely
    elif any(i in ai.text for i in ["thank","thanks"]):
        res = np.random.choice(["you're welcome!","anytime!","no problem!","cool!","I'm here if you need me!","mention not"])

    elif any(i in ai.text for i in ["exit","close"]):
        res = np.random.choice(["Tata","Have a good day","Bye","Goodbye","Hope to meet soon","peace out!"])
```

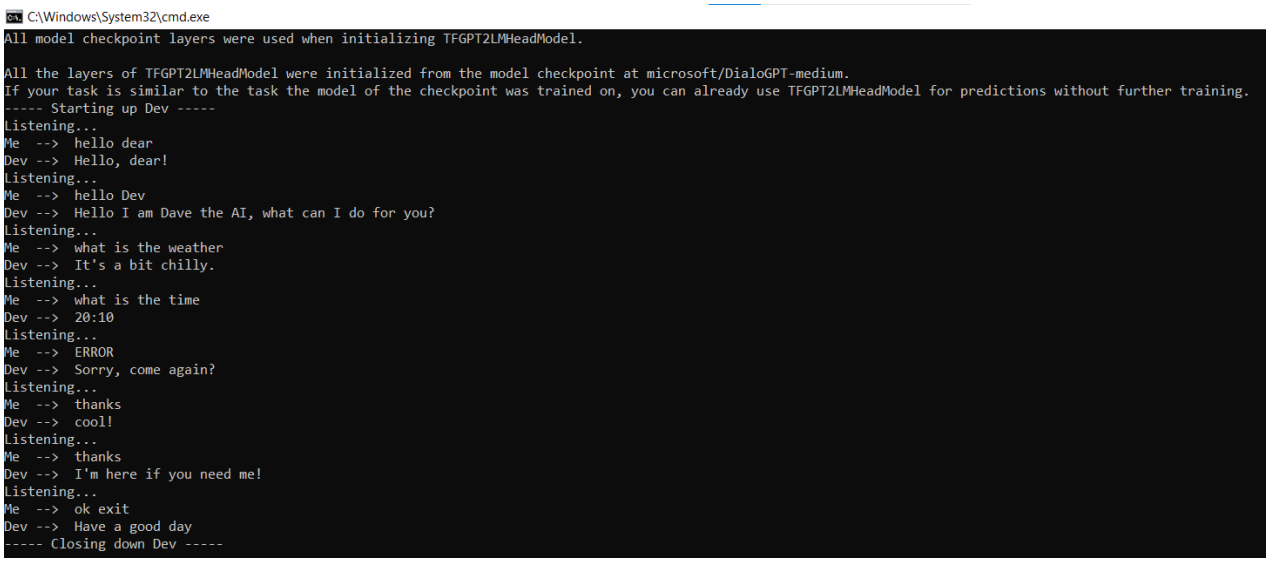
```
        ex=False
    ## conversation
    else:
        if ai.text=="ERROR":
            res="Sorry, come again?"
        else:
            chat = nlp(transformers.Conversation(ai.text), pad_token_id=50256)
            res = str(chat)
            res = res[res.find("bot >> ")+6:].strip()


    ai.text_to_speech(res)
print("----- Closing down Dev -----")
```

CHAPTER 4

DESIRED RESULT

```
----- starting up Dev -----
listening...
me --> Hey Dev
AI --> Hello I am Dev the AI, what can I do for you?
listening...
me --> What is the time
AI --> 17:30
listening...
me --> thanks
AI --> cool!
listening...
```



CONCLUSION

we have demonstrated a step-by-step tutorial that you can utilize to create a conversational Chatbot. This chatbot can be further enhanced to listen and reply as a human would. The codes included here can be used to create similar chatbots and projects. To conclude, we have used Speech Recognition tools and NLP tech to cover the processes of text to speech and vice versa. Pre-trained Transformers language models were also used to give this chatbot intelligence instead of creating a scripted bot. Now, you can follow along or make modifications to create your own chatbot or virtual assistant to integrate into your business, project, or your app support functions. Thanks for reading and hope you have fun recreating this project.

REFERENCES

- Head First Python: by Paul Barry
- Python Programming :by John Zelle
- Python : A Guide for Data Scientists
- Python for Data Analysis
- Digital Skills by Future Learn
- Google
- Artificial Intelligence for Humans -by Jeff Heaton
- AI:The Basics-by KevinWarwick
- Head First Python: by Paul Barry
- Python Programming :by John Zelle
- Python : A Guide for Data Scientists
- Python for Data Analysis