Artificial Intelligence Laboratory Manual

CSCL-4101
Semester 6th

Name: _____

Roll Number: _____

Artificial Intelligence BS (CS)
Shaheed Zulfikar Ali Bhutto Institute of Science and Technology
(SZABIST)

List of Experiments

Lab No	Topics	Remarks	Mapping
1.	Introduction to Python Programming and to get familiar with Integrated Development Environment (IDE), Google Colab online laboratory, Jupyter notebook. Python fundamentals and Introduction to AI and Algorithms.		CLO_1
2.	Basic Programming in Python: Data Types and Variables, Types of Operators and Operator Precedence, Advance data types, List/tuple/dictionaries/slicing.		CLO_1
3.	Python: Conditional statements (If-Else, Nested if-else and if-elif-else ladder), Understanding of Control Structure: For Loop, While Loop), Concept of Functions (built-in, user defined functions, Recursion), Concept of Arrays, Structures.		CLO_2
4.	Familiarization with Python packages for AI: Pandas, Numpy, Sci-kit learn, Mat-plot library. Machine Learning: Artificial Neural Network (Single Layer Perceptron)		CLO_2
5.	Machine Learning: Artificial Neural Network (Multi Layer Perceptron)		CLO_2
6.	Machine Learning: K-Nearest Neighbor KNN.		CLO_2
7.	Machine Learning: Correlation, Regression		CLO_2
8.	Machine Learning: K-mean clustering		CLO_2
9.	Mid Lab		CLO_3
10.	Machine Learning: Decision Tree		CLO_2
11.	Machine Learning: Naïve Bayes		CLO_2
12.	Machine Learning: SVM Apriori Algorithm		CLO_2
13.	Optimization: Genetic Algorithm GA		CLO_2
14.	Solution search strategies :Breath-first search, Depth first search, Best First search		CLO_2
15.	Project		CLO_3
16.	Final Lab		CLO_3

CS, SZABIST University

CSCL-4101 Artificial Intelligence, Spring 2022
Instructor: **Mr. Usama Khalid** Office Hours: 3:00PM – 06:00PM (Monday -Friday)

Prerequisites: **Programming fundamentals**

Objectives:
The goals of the Artificial Intelligence laboratory are;

1. To allow the students to perform experiments that demonstrate the theory of artificial intelligence that is discussed in the theory course.
2. To introduce the basic principles, techniques, and applications of Artificial Intelligence.
3. More emphasis will be placed on the implementation of AI algorithm on programming tools.
4. Assigned projects promote a 'hands-on' approach for understanding, as well as a challenging avenue for exploration and creativity. Specifically: a. Become familiar with basic principles of AI toward problem solving, inference, perception, knowledge representation, and learning. b. Investigate applications of AI techniques in intelligent agents, expert systems, artificial neural networks and other machine learning models. c. Experience AI development tools such as an 'AI language', expert system shell, and/or data mining tool. d. Experiment with a machine learning model for simulation and analysis. e. Explore the current scope, potential, limitations, and implications of intelligent systems.

Contents:
Introduction to Python Programming, Integrated Development Environment (IDE), Google Colab online laboratory, Jupyter notebook. Introduction to AI and Algorithms, Basic Programming in Python: Data Types and Variables, Types of Operators and Operator Precedence, Advance data types, List/tuple/dictionaries/slicing, Conditional statements (If-Else, Nested if-else and if-elif-else ladder), Understanding of Control Structure: For Loop, While Loop), Concept of Functions (built-in, user defined functions, Recursion), Concept of Arrays, Structures, Familiarization with Python packages for AI: Pandas, Numpy, Sci-kit learn, Mat-plot library, Machine Learning: Artificial Neural Network (Single Layer Perceptron, Multi Layer Perceptron, KNN, K-mean clustering, Decision Tree, Regression, Naïve Bayes, SVM Apriori Algorithm , market basket analysis, Principal Component Analysis PCA. Optimization: Genetic Algorithm GA, Particle Swarm Optimization PSO. Solution search strategies :Breath-first search, Depth first search, Best First search, Beam search, Bi-directional search, Uniform cost search, Iterative Deepening Depth First search, Informed search, Heuristic Search. Deep Learning: Deep Network Designing, Transfer Learning.

Learning Outcomes

Mapping of CLOs and PLOs			
S. No.	**Course Learning Outcomes**	**PLOs**	**Bloom's Taxonomy**
CLO_1	Be able to **practice** measuring variables of different physical systems By the end of this lesson, the student will be able to describe and **understand** the basics of AI and python programming interface.	PLO_1	C2 (Understand)
CLO_2	Be able to **implement** the algorithms and techniques in artificial intelligence	PLO_2	C3 (Apply)
CLO_3	Be able to **experiment and analyze** the algorithms and techniques in artificial intelligence	PLO_3	C4 (Analyze)

CLO Assessment Mechanism				
	Assessment tools	**CLO_1**	**CLO_2**	**CLO_3**
	Lab Performance	30%	40%	30%
	Lab Assignment/Quiz	70%	30%	
	Final Examination	20%	60%	20%

Overall Grading Policy	
Assessment Items	**Percentage**
Lab Performance	15%
Lab Manual	15%
Mid Examination	20%
Final Examination	30%
Lab Project	20%

Recommended Book:

- Artificial Intelligence. A Modern Approach, Stuart Russell and Peter Norvig, null Edition, Prentice Hall
- Understanding Machine Learning: From Theory to Algorithms, Shai Shalev-Shwartz, Shai Ben-David, null Edition, Cambridge University Press,2020

Administrative Instructions:

- Title and Group members name for Lab/Course project should be submitted by 8th week of lab.
- According to institute policy, 75% attendance is *mandatory* to appear in the final examination but 100% will be expected. Approved leaves will not be considered towards attendance.
- Every student is expected to be in lab before schedule starting time.
- In any case there will be no rescheduling and makeup of labs.
- To maintain courtesy and civility, you may not use inappropriate or offensive commentary or body language to show your attitude regarding the course, the instructor, assignments, or fellow students. Profanity and Offensive Language: You may not use profanity or offensive language in class.
- You may not work on other activities while in lab. This includes homework for other courses or other personal activities. Internet: Yyou may use the Internet for valid, academic purposes only. You may not use it for open access to other non-academic sites which are unrelated to the course.
- Cell phones: You may not receive or send telephone calls during class and must refrain from using your phone for browsing, games, texting or other purposes during class time. You are responsible for turning off and putting your phones away upon entering class.

Marks Evaluation

Experiment No.	Marks			
	Class Participation (0.3)	**Experiment Performance (0.5)**	**Experiment Reporting (0.2)**	**Total (1)**
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
Total				

Instructor's signature

Lab Semester Project Rubrics

Heads	**Marks** (*m*)	**Performance below Poor** (0% of m)	**Poor** (20% of *m*)	**Performance between Poor and Satisfactory** (40% of *m*)	**Satisfactory** (60% of *m*)	**Performance between Satisfactory and Excellent** (80% of *m*)	**Excellent** (100% of *m*)
Understanding of code and algorithm	6						
Communication skills	3						
Viva	5						
Presentation	3						
Report	3						
Total Marks	20						
Marks Obtained							

Introduction to Python, Artificial Intelligence and to get familiar with Integrated Development Environment (IDE)

Objective:

To discuss and elaborate AI and algorithms, python programming fundamentals and get familiarization with Integrated Development Environment (IDE), Google Colab online laboratory, Jupyter notebook.

Software and Tools:

- 1.6 GHz or faster processor
- 4 GB of RAM
- 20 GB of available hard disk space
- Windows 7 or later
- Spyder (Python 3.9)
- Google Colab

Background and Procedure:

Introduction to Artificial Intelligence

Artificial intelligence (AI) is a wide-ranging branch of computer science concerned with building smart machines capable of performing tasks that typically require human intelligence.
"Artificial intelligence is a set of algorithms and intelligence to try to mimic human intelligence. Machine learning is one of them, and deep learning is one of those machine learning techniques."

Types of Artificial Intelligence

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

Examples of AI Applications

- Siri, Alexa and other smart assistants
- Self-driving cars (Driver-less Cars)
- Robo-advisors
- Conversational bots (Chat Bots)
- Email spam filters
- Netflix's recommendations (Recommendation system)

The major limitation in defining AI as simply "building machines that are intelligent" is that it doesn't actually explain what artificial intelligence is? What makes a machine intelligent? AI is an interdisciplinary science with multiple approaches, but advancements in **machine learning** and **deep learning** are creating a paradigm shift in virtually every sector of the tech industry.

Categories of Artificial Intelligence
Artificial intelligence generally falls under two broad categories:

1. **Narrow AI:**

Sometimes referred to as "Weak AI," this kind of artificial intelligence operates within a limited context and is a simulation of human intelligence. Narrow AI is often focused on performing a single task extremely well and while these machines may seem intelligent, they are operating under far more constraints and limitations than even the most basic human intelligence.

Example:

- Google search
- Image recognition software
- Siri, Alexa and other personal assistants
- Self-driving cars
- IBM's Watson

2. **Artificial General Intelligence (AGI):**

AGI, sometimes referred to as "Strong AI," is the kind of artificial intelligence we see in the movies, like the robots from Westworld or Data from Star Trek: The Next Generation. AGI is a machine with general intelligence and, much like a human being, it can apply that intelligence to solve any problem.

Reactive Machines
A reactive machine follows the most basic of AI principles and, as its name implies, is capable of only using its intelligence to perceive and react to the world in front of it. A reactive machine cannot store a memory and as a result cannot rely on past experiences to inform decision making in real-time. Perceiving the world directly means that reactive machines are designed to complete only a limited number of specialized duties.
A famous example of a reactive machine is **Deep Blue**, which was designed by IBM in the 1990's as a chess-playing supercomputer and defeated international grandmaster Gary Kasparov in a game.
Another example of a game-playing reactive machine is Google's **AlphaGo**. **AlphaGo** is also incapable of evaluating future moves but relies on its own neural network to evaluate developments of the present game, giving it an edge over Deep Blue in a more complex game. **AlphaGo** also bested world-class competitors of the game, defeating champion Go player Lee Sedol in 2016.

Limited Memory

Limited memory artificial intelligence has the ability to store previous data and predictions when gathering information and weighing potential decisions — essentially looking into the past for clues on what may come next. Limited memory artificial intelligence is more complex and presents greater possibilities than reactive machines.

Limited memory AI is created when a team continuously trains a model in how to analyze and utilize new data or an AI environment is built so models can be automatically trained and renewed. When utilizing limited memory AI in machine learning, six steps must be followed: Training data must be created, the machine learning model must be created, the model must be able to make predictions, the model must be able to receive human or environmental feedback, that feedback must be stored as data, and these steps must be reiterated as a cycle.

There are three major machine learning models that utilize limited memory artificial intelligence:

- **Reinforcement learning** - Which learns to make better predictions through repeated trial-and-error.

- **Long Short Term Memory (LSTM) -** Which utilizes past data to help predict the next item in a sequence. LTSMs view more recent information as most important when making predictions and discounts data from further in the past, though still utilizing it to form conclusions.

- **Evolutionary Generative Adversarial Networks (E-GAN)** - Which evolves over time, growing to explore slightly modified paths based off of previous experiences with every new decision. This model is constantly in pursuit of a better path and utilizes simulations and statistics, or chance, to predict outcomes throughout its evolutionary mutation cycle.

Theory of Mind:

Theory of Mind is just that **theoretical**. We have not yet achieved the technological and scientific capabilities necessary to reach this next level of artificial intelligence.

The concept is based on the psychological premise of understanding that other living things have thoughts and emotions that affect the behavior of one's self. In terms of AI machines, this would mean that AI could comprehend how humans, animals and other machines feel and make decisions through self-reflection and determination, and then will utilize that information to make decisions of their own. Essentially, machines would have to be able to grasp and process the concept of "mind," the fluctuations of emotions in decision making and a litany of other psychological concepts in real time, creating a two-way relationship between people and artificial intelligence.

Self-Awareness:

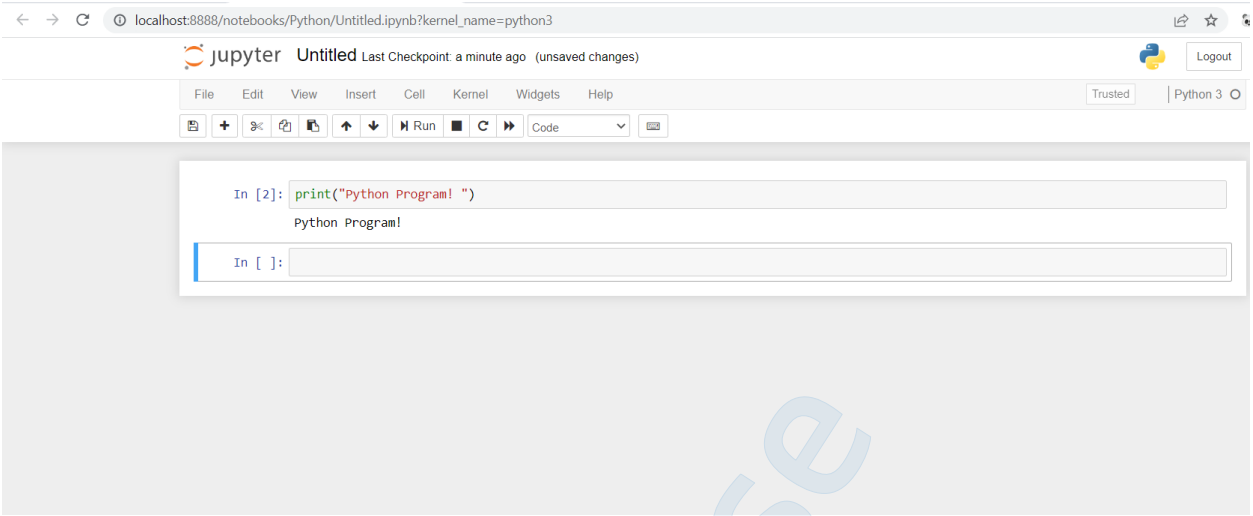
Once Theory of Mind can be established in artificial intelligence, sometime well into the future, the final step will be for AI to become self-aware. This kind of artificial intelligence possesses human-level consciousness and understands its own existence in the world, as well as the presence and emotional state of others. It would be able to understand what others may need based on not just what they communicate to them but how they communicate it.
Self-awareness in artificial intelligence relies both on human researchers understanding the premise of consciousness and then learning how to replicate that so it can be built into machines.

Introduction to Python
Python is an open source and cross-platform programming language that has become increasingly popular over the last ten years. It was released in 1991. It is an interpreted, object-oriented, high-level programming language with dynamic semantics developed by Guido van Rossum. Latest version is 3.7.0. Python is a multi-purpose programming languages (due to its many extensions).
Python is an interpreted programming language, this means that as a developer you write Python (.py) files in a text editor and then put those files into the python interpreter to be executed. Depending on the Editor you are using, this is either done automatically, or you need to do it manually.

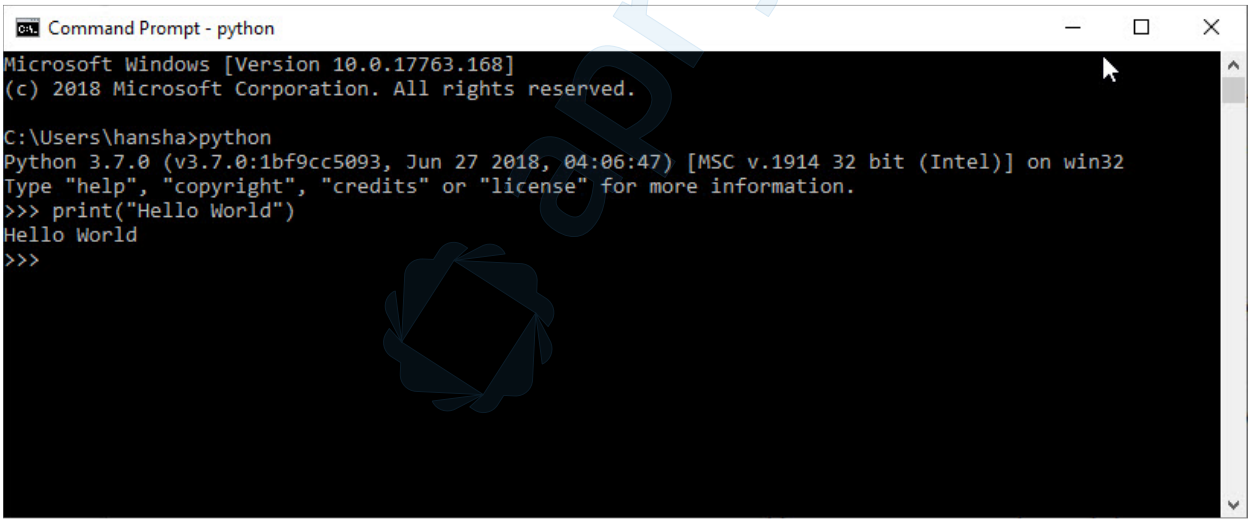
Examples: scientific computing and calculations, simulations, web development (Django Web framework), etc.

Introduction to Anaconda:
Anaconda offers the easiest way to perform Python/R data science and machine learning on a single machine. You can work with thousands of open-source packages and libraries with it. Anaconda is a distribution package, where you get Python compiler, Python packages and the Spyder editor. Anaconda includes Python, Jupyter Notebook, and other commonly used packages for scientific computing and data science.

Jupyter Notebook:
The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and text.



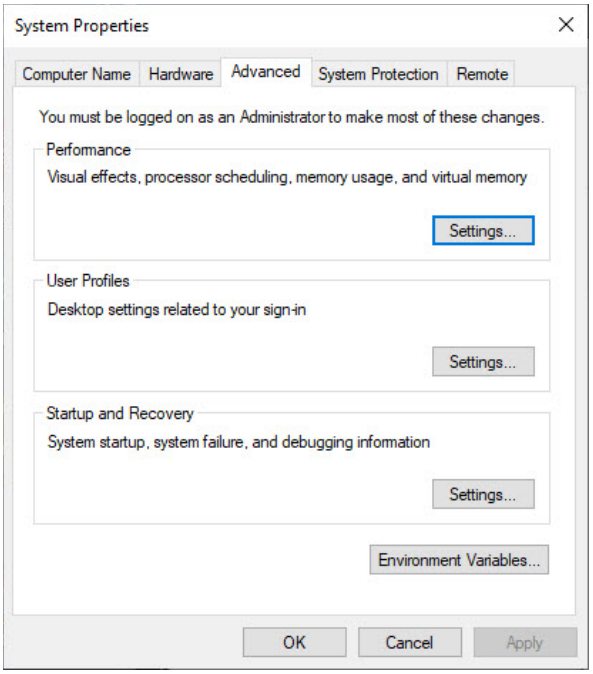
Lets open your Python Editor and type the following:
Print("Hello World")



If you get an error message like this:
'python' is not recognized as an internal or external command, operable program or batch file.
Then you need to add Python to your path. See instructions below.

Add Python to Path:

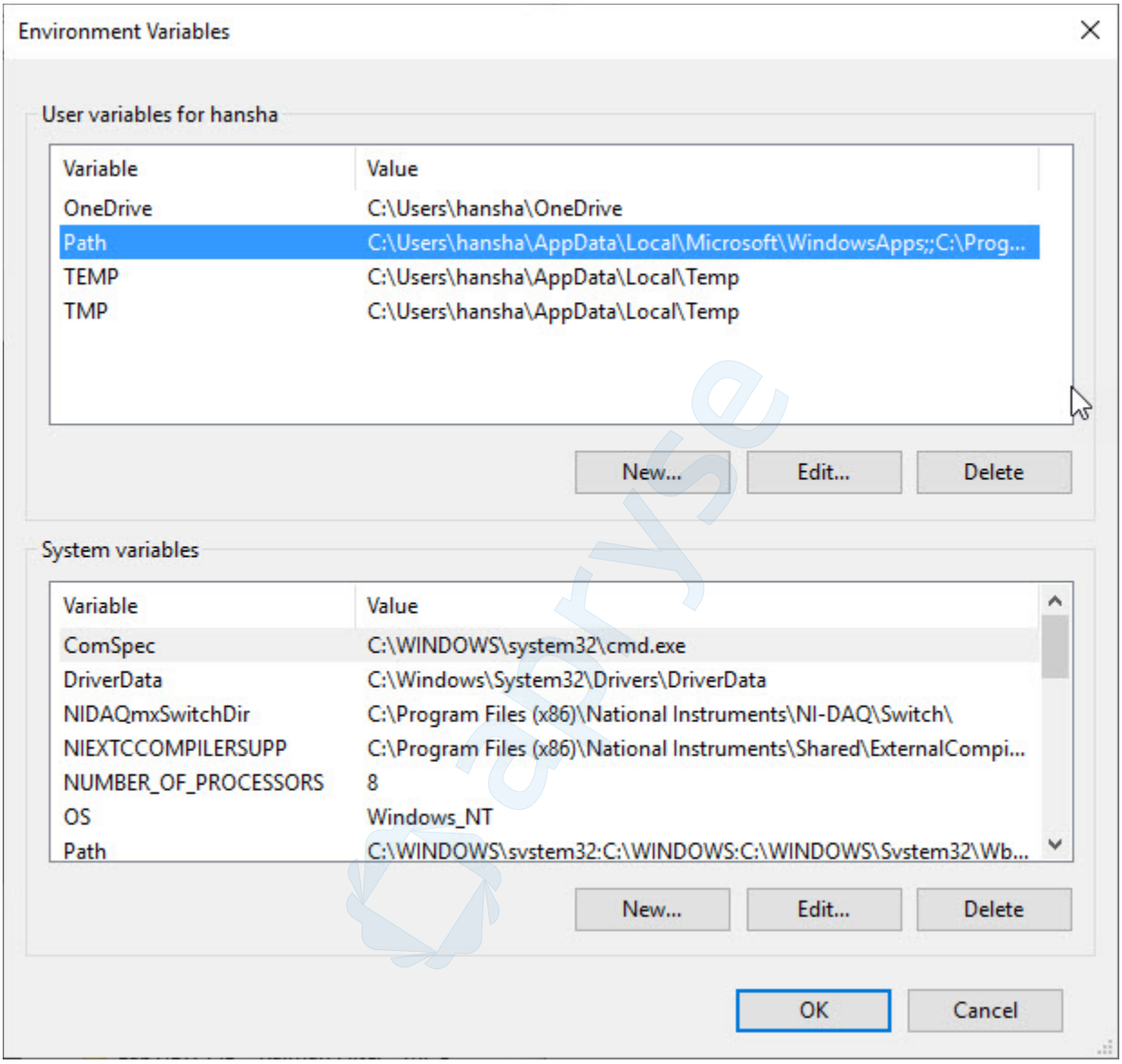
In the Windows menu, search for \advanced system settings" and select View advanced system settings.
In the window that appears, click Environment Variables . . . near the bottom right.

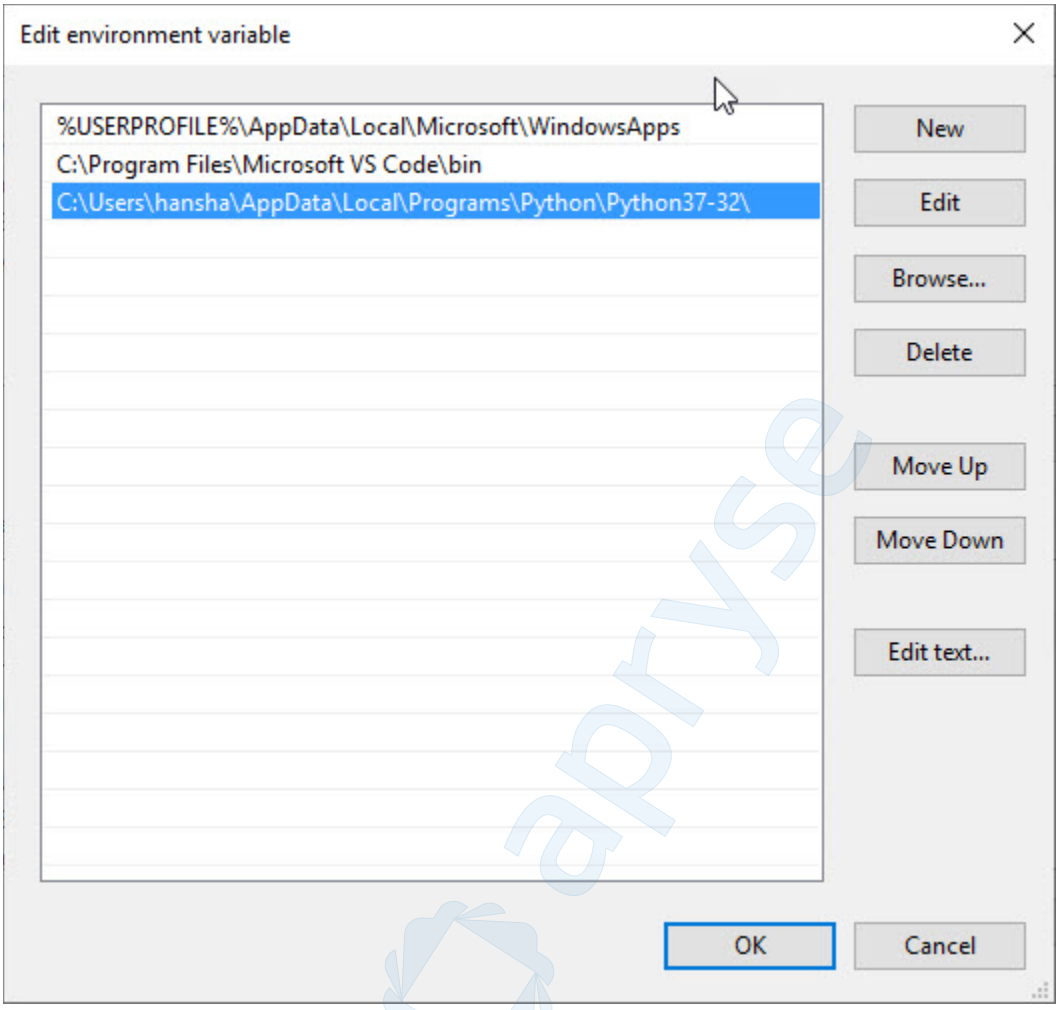


In the next window, select the user variable named Path and click Edit. . . to change its value. Select "New" and add the path where "python.exe" is located. See Figure 3.6.
The Default Location is:

```
C:\Users\user\AppData\Local\Programs\Python\Python37-32\
```

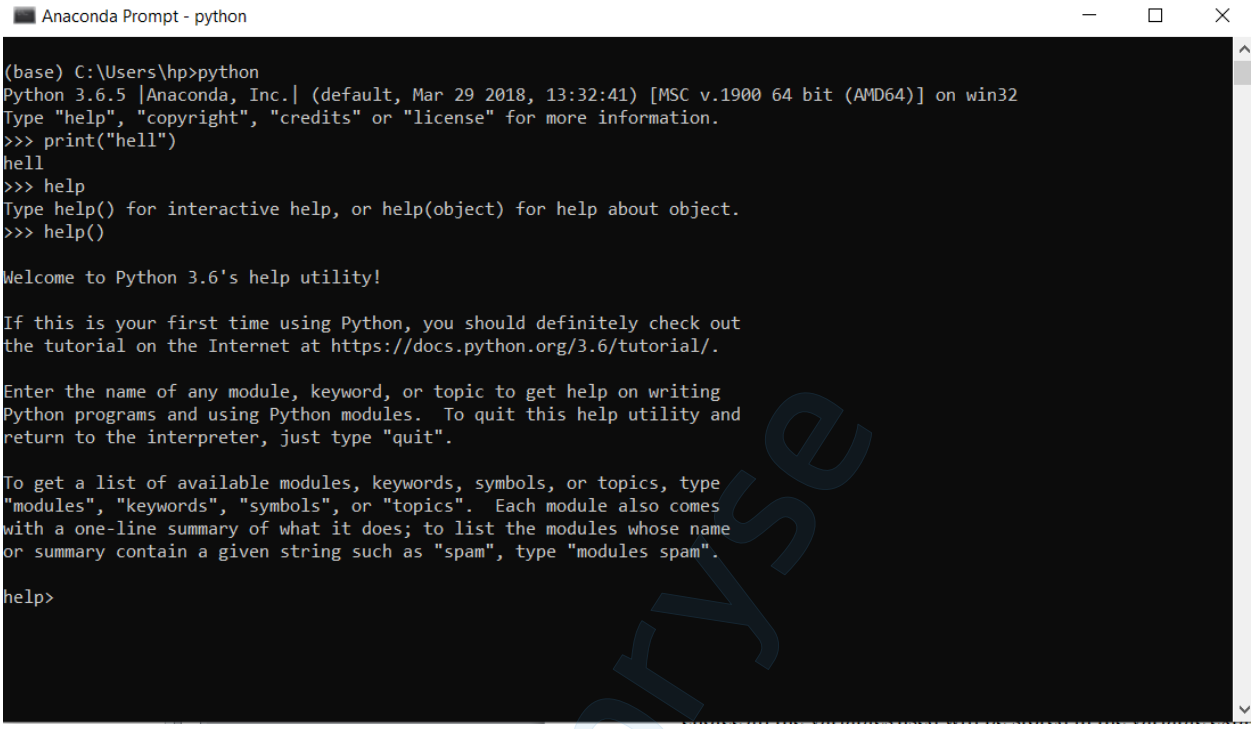
Click Save and open the Command Prompt once more and enter "python" to verify it works.

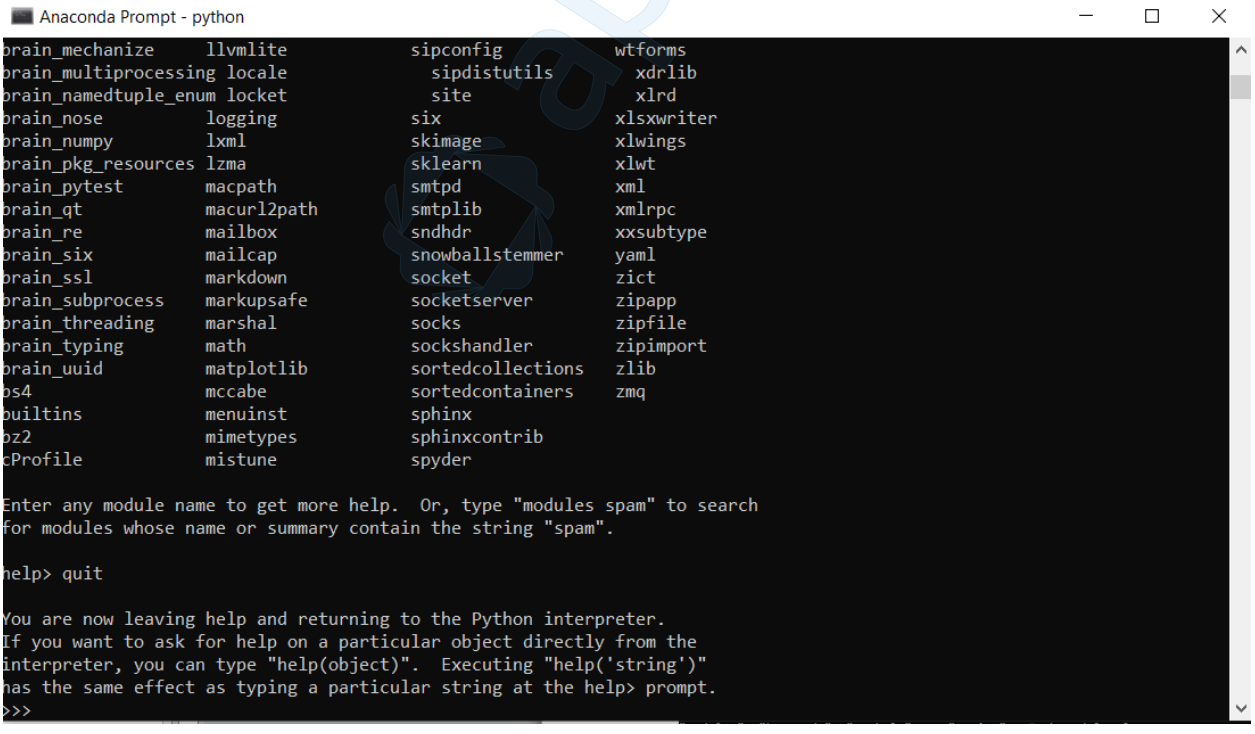

Environment Variables
User variables for hansha
Variable
Value
OneDrive
C:\Users\hansha\OneDrive
Path
C:\Users\hansha\AppData\Local\Microsoft\WindowsApps;;C:\Prog...
TEMP
C:\Users\hansha\AppData\Local\Temp
TMP
C:\Users\hansha\AppData\Local\Temp
New...
Edit...
Delete
System variables
Variable
Value
ComSpec
C:\WINDOWS\system32\cmd.exe
DriverData
C:\Windows\System32\Drivers\DriverData
NIDAQmxSwitchDir
C:\Program Files (x86)\National Instruments\NI-DAQ\Switch\
NIEXTCCOMPILERSUPP
C:\Program Files (x86)\National Instruments\Shared\ExternalCompi...
NUMBER_OF_PROCESSORS
8
OS
Windows_NT
Path
C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wb...
New...
Edit...
Delete
OK
Cancel



Get Help:

An extremely useful command is help(), which enters a help functionality to explore all the students python lets you do, right from the interpreter. Press q to close the help window and return to the Python prompt.


Anaconda Prompt - python
(base) C:\Users\hp>python
Python 3.6.5 |Anaconda, Inc.| (default, Mar 29 2018, 13:32:41) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("hell")
hell
>>> help
Type help() for interactive help, or help(object) for help about object.
>>> help()
Welcome to Python 3.6's help utility!
If this is your first time using Python, you should definitely check out
the tutorial on the Internet at https://docs.python.org/3.6/tutorial/.
Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules. To quit this help utility and
return to the interpreter, just type "quit".
To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics". Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".
help>


Anaconda Prompt - python
brain_mechanize llvmlite sipconfig wtforms
brain_multiprocessing locale sipdistutils xdrlib
brain_namedtuple_enum locket site xlrd
brain_nose logging six xlsxwriter
brain_numpy lxml skimage xlwings
brain_pkg_resources lzma sklearn xlwt
brain_pytest macpath smtpd xml
brain_qt macurl2path smtplib xmlrpc
brain_re mailbox sndhdr xxsubtype
brain_six mailcap snowballstemmer yaml
brain_ssl markdown socket zict
brain_subprocess markupsafe socketserver zipapp
brain_threading marshal socks zipfile
brain_typing math sockshandler zipimport
brain_uuid matplotlib sortedcollections zlib
bs4 mccabe sortedcontainers zmq
builtins menuinst sphinx
bz2 mimetypes sphinxcontrib
cProfile mistune spyder
Enter any module name to get more help. Or, type "modules spam" to search
for modules whose name or summary contain the string "spam".
help> quit
You are now leaving help and returning to the Python interpreter.
If you want to ask for help on a particular object directly from the
interpreter, you can type "help(object)". Executing "help('string')"
has the same effect as typing a particular string at the help> prompt.
>>>

Introduction to Spyder:

There are various IDEs in the market to select from such as Spyder, Atom, Pycharm, Pydev etc. Data scientists prefer Spyder among all the different IDEs available and the driving fact behind this is that Spyder was built specifically to be used for data science. Its interface allows the user to scroll through various data variables and also ready to use online help option. The output of the code can be viewed in the python console on the same screen. You can work on different scripts at a moment and then try them out one by one in the same console or different as per your choice all the variables used will be stored in the variable explorer tab. It also provides an option to view graphs and visualizations in the plot window. You can also cover the basics concepts by taking up free Syder python and also check out Python Libraries for Machine Learning from Great Learning Academy.

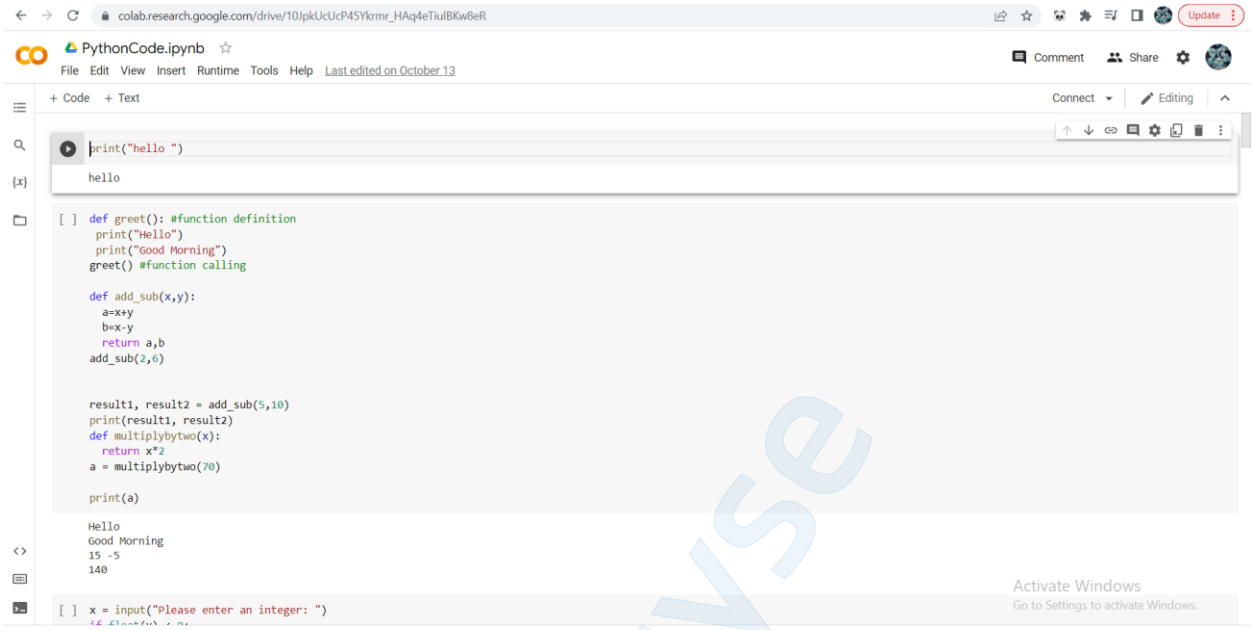
Download and installation:

Go to Anaconda Website: https://www.anaconda.com/. Download anaconda for your respective operating system. Download and install Anaconda.
You can launch Spyder with anaconda navigator.

Google Colab

Colaboratory, or "Colab" for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. More technically, Colab is a hosted Jupyter notebook service that requires no setup to use, while providing access free of charge to computing resources including GPUs.

Colab Interface



Limitations

Resources in Colab are prioritized for interactive use cases. We prohibit actions associated with bulk compute, actions that negatively impact others, as well as actions associated with bypassing our policies. The following are disallowed from Colab runtimes:

- file hosting, media serving, or other web service offerings not related to interactive compute with Colab
- downloading torrents or engaging in peer-to-peer file-sharing
- using a remote desktop or SSH
- connecting to remote proxies
- mining cryptocurrency
- running denial-of-service attacks
- password cracking
- using multiple accounts to work around access or resource usage restrictions
- creating deepfakes

Basic Programming in Python

Objective:

To perform the basic python programming using Data Types and Variables, Types of Operators and Operator Precedence, Advance data types, List/tuple/dictionaries/slicing.

Background and Procedure:

Python Data Types

Standard data types

With Python you don't get so much out of the box. Instead of having all of its functionality built into its core, you need to install different packages for different topics.
The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

- Numeric data types: int, float, complex
- String data types: str
- Sequence types: list, tuple, range
- Binary types: bytes, bytearray, memoryview
- Mapping data type: dict
- Boolean type: bool
- Set data types: set, frozenset

Python Numeric Data types

var1 = 10
var2 = 2.55
Python supports three different numerical types −

- int (signed integers)
- float (floating point real values)
- complex (complex numbers)

```
#int (signed integers)
var1 = 10
#float (floating point real values)
var2 = 36.569
#complex (complex numbers)
var3 = 6+2j
```

All integers in Python 3 are represented as long integers. Hence, there is no separate number type as long.

int	float	complex
10	0.0	3.14j
100	15.20	45.j
-786	-21.9	9.322e-36j
080	32.3+e18	.876j
-0490	-90.	-.6545+0J
-0x260	-32.54e100	3e+26J
0x69	70.2-E12	4.53e-7j

Python Strings

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows either pair of single or double quotes. Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 to the end. The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

String Input:

Python allows for command line input. That means we are able to ask the user for input.
The following example asks for the user's name, then, by using the **input()** method, the program prints the name to the screen:

```
#input string

print("enter value")
value = input()
print(value)
```

```
enter value
demo
```

```
#input string

print("enter value")
value = input()
print("the value is", value)
```

```
enter value
demo
the value is demo
```

```
str= "Artificial Inyelligence Lab"
print (str) # Prints complete string
print (str[0]) # Prints first character of the string
print (str[2:5]) # Prints characters starting from 3rd to 5th
print (str[2:]) # Prints string starting from 3rd character
print (str * 2) # Prints string two times
print (str + "TEST") # Prints concatenated string
```

```
Artificial Inyelligence Lab
A
tif
tificial Inyelligence Lab
Artificial Inyelligence LabArtificial Inyelligence Lab
Artificial Inyelligence LabTEST
```

Python

Lists

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One of the differences between them is that all the items belonging to a list can be of different data type.

The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

```
#accessing different values stored in array
list = [ 'python', 123 , 2.663, 'code', 10.2 ]
tinylist = [123, 'john']
print (list) # Prints complete list
print (list[0]) # Prints first element of the list
print (list[1:3]) # Prints elements starting from 2nd till 3rd
print (list[2:]) # Prints elements starting from 3rd element
print (tinylist * 2) # Prints list two times
print (list + tinylist) # Prints concatenated lists
```

```
['python', 123, 2.663, 'code', 10.2]
python
[123, 2.663]
[2.663, 'code', 10.2]
[123, 'john', 123, 'john']
['python', 123, 2.663, 'code', 10.2, 123, 'john']
```

Python Tuples

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parenthesis.
The main difference between lists and tuples is- Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be thought of as read-only lists.

The following code is invalid with tuple, because we attempted to update a tuple, which is not allowed. Similar case is possible with lists:

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tuple[2] = 1000 # Invalid syntax with tuple
list[2] = 1000 # Valid syntax with list
```

```
tuple = ( 'python', 123 , 2.663, 'code', 10.2 )
subtuple = (999, 'program')
print (tuple) # Prints complete tuple
print (tuple[0]) # Prints first element of the tuple
print (tuple[1:3]) # Prints elements starting from 2nd till 3rd
print (tuple[2:]) # Prints elements starting from 3rd element
print (subtuple * 2) # Prints tuple two times
print (tuple + subtuple) # Prints concatenated tuple
```

```
('python', 123, 2.663, 'code', 10.2)
python
(123, 2.663)
(2.663, 'code', 10.2)
(999, 'program', 999, 'program')
('python', 123, 2.663, 'code', 10.2, 999, 'program')
```

Python Dictionary

Python's dictionaries are kind of hash-table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any
Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.
Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).
Dictionaries have no concept of order among the elements. It is incorrect to say that the elements are "out of order"; they are simply unordered.

```
#Access all the keys and values inside the structured dictionary.

dict = {}
dict['one'] = "This is first"
dict[2] = "This is second"
smalldict = {'name': 'zendra','code':6734, 'dept': 'IT'}
print (dict['one']) # Prints value for 'one' key
print (dict[2]) # Prints value for 2 key
print (smalldict) # Prints complete dictionary
print (smalldict.keys()) # Prints all the keys
print (smalldict.values()) # Prints all the values
```

```
This is first
This is second
{'name': 'zendra', 'code': 6734, 'dept': 'IT'}
dict_keys(['name', 'code', 'dept'])
dict_values(['zendra', 6734, 'IT'])
```

Python Range() function

We can generate a sequence of numbers using range() function. range(10) will generate numbers from 0 to 9 (10 numbers).
We can also define the start, stop and step size as range(start, stop,step_size). step_size defaults to 1 if not provided.
This function does not store all the values in memory; it would be inefficient. So it remembers the start, stop, step size and generates the next number on the go.

```
#range function

range(5) #[0,1,2,3,4]
range(1,5) #[1,2,3,4]
range(1,10,2) #[1,3,5,7,9]

print(range(5))

for i in range(0, 5):
    print (i)
else:
    print ('The range is over')
```

```
range(0, 5)
0
1
2
3
4
The range is over
```

```
#this Range function program executes and prints the sum of the numbers entered by user
Sum = 0
print("Please Enter 10 Numbers\n")
for i in range(1, 11):
    num = int(input("Number %d = " %i))
    Sum = Sum + num

avg = Sum / 10

print("The Sum of 10 Numbers     = ", Sum)
print("The Average of 10 Numbers = ", avg)
```

```
Please Enter 10 Numbers

Number 1 = 45
Number 2 = 56
Number 3 = 89
Number 4 = 55
Number 5 = 2
Number 6 = 55
Number 7 = 33
Number 8 = 5
Number 9 = 66
Number 10 = 78
The Sum of 10 Numbers     =  484
The Average of 10 Numbers =  48.4
```

Python Slice() function

A slice object is used to specify how to slice a sequence. You can specify where to start the slicing, and where to end. You can also specify the step, which allows you to e.g. slice only every other item.
Following is the syntax:

- *slice(start, end, step)*
- *slice (stop)*
- *slice (start, stop[, step])*

Following are the Parameters:

- *start: Starting index where the slicing of object starts.*
- *stop: Ending index where the slicing of object stops.*
- *step: It is an optional argument that determines the increment between each index for slicing.*

Return Type: Returns a sliced object containing elements in the given range only.

A sequence of objects of any type (string, bytes, tuple, list or range) or the object which implements __getitem__() and __len__() method then this object can be sliced using slice() method.

```
# Python slice() function
tup = (45,68,955,1214,41,558,636,66)
slic = slice(0,10,3) # returns slice object
slic2 = tup[0:10:3] # fetch the same elements
# We can use this slice object to get elements
str2 = tup[slic]
# Displaying result
print(str2)
print(slic2)
```

```
(45, 1214, 636)
(45, 1214, 636)
```

```
String = 'Hello World'
slice_obj = slice(5,11)
print(String[slice_obj])
```

```
 World
```

```
values = ("a", "b", "c", "d", "e", "f", "g", "h")
x = slice(3, 5)
print(values[x])
```

```
('d', 'e')
```

Python Variables

Variables are nothing but reserved memory locations to store values. It means that when you create a variable, you reserve some space in the 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 the variables, you can store integers, floats or characters in these variables.
Variables are defined with the assignment operator, \=". Python is dynamically typed, meaning that variables can be assigned without declaring their type, and that their type can change. Values can come from constants, from computation involving values of other variables, or from the output of a function.

Basic rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters (A-z, 0-9) and underscores
- Variable names are case-sensitive, e.g., amount, Amount and AMOUNT are three different variables.

Assigning Values to Variables

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.
The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable.

```
#Variables

#Initializing numeric type datatype variables.
num = 100 # An integer assignment
miles = 12.0 # A floating point
name = "Sam" # A string
print (num)
print (miles)
print (name)
```

```
100
12.0
Sam
```

Multiple Assignments

Python allows you to assign a single value to several variables simultaneously.

For example
a = b = c = 1
Here, an integer object is created with the value 1, and all the three variables are assigned to the same memory location. You can also assign multiple objects to multiple variables.

```
#python variables
#Multiple assignments
a = b = c = 1
print(a)
print(b)
print(c)
```

```
1
1
1
```

For example
a, b, c = 1, 2, "john"
Here, two integer objects with values 1 and 2 are assigned to the variables a and b respectively, and one string object with the value "john" is assigned to the variable c.

```
a, b, c = 1, 2, "john"

print(a)
print(b)
print(c)
```

```
1
2
john
```

Types of Operators

Python language supports the following types of operators:

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Python Arithmetic Operators

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 31
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -11
* Multiplication	Multiplies values on either side of the operator	a * b = 210
/ Division	Divides left hand operand by right hand operand	b / a = 2.1
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 1
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed.	9//2 = 4 and 9.0//2.0 = 4.0

Example Program:
Assume variable a holds 50 and variable b holds 23:

```
#Arithmatic Operators
a = 50
b = 23
c = 0
c = a + b
print ("Line 1 - Value of c is ", c)

c = a - b
print ("Line 2 - Value of c is ", c )

c = a * b
print ("Line 3 - Value of c is ", c)

c = a / b
print ("Line 4 - Value of c is ", c )

c = a % b
print ("Line 5 - Value of c is ", c)

c = a**b
print ("Line 6 - Value of c is ", c)

c = a//b
print ("Line 7 - Value of c is ", c)
```

```
Line 1 - Value of c is  73
Line 2 - Value of c is  27
Line 3 - Value of c is  1150
Line 4 - Value of c is  2.1739130434782608
Line 5 - Value of c is  4
Line 6 - Value of c is  11920928955078125000000000000000000000000
Line 7 - Value of c is  2
```

Python Comparison Operators

These operators compare the values on either side of them and decide the relation among them. They are also called Relational operators.

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a!= b) is true.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Example Program:

```python
#Python Comparison Operators
a = 21
b = 10
if ( a == b ):
    print ("Line 1 - a is equal to b")
else:
    print ("Line 1 - a is not equal to b")
if ( a != b ):
    print ("Line 2 - a is not equal to b")
else:
    print ("Line 2 - a is equal to b")
if ( a < b ):
    print ("Line 3 - a is less than b" )
else:
    print ("Line 3 - a is not less than b")
if ( a > b ):
    print ("Line 4 - a is greater than b")
else:
    print ("Line 4 - a is not greater than b")
a,b=b,a #values of a and b swapped. a becomes 10, b becomes 21
if ( a <= b ):
    print ("Line 5 - a is either less than or equal to b")
else:
    print ("Line 5 - a is neither less than nor equal to b")
if ( b >= a ):
    print ("Line 6 - b is either greater than or equal to b")
else:
    print ("Line 6 - b is neither greater than nor equal to b")
```

```
Line 1 - a is not equal to b
Line 2 - a is not equal to b
Line 3 - a is not less than b
Line 4 - a is greater than b
Line 5 - a is either less than or equal to b
Line 6 - b is either greater than or equal to b
```

Python Assignment Operators

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / ac /= a is equivalent to c = c / a
%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor Division	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

Example Program:

```
#Python Assignment Operators
a = 21
b = 10
c = 0
c = a + b
print ("Line 1 - Value of c is ", c)
c += a
print ("Line 2 - Value of c is ", c )
c *= a
print ("Line 3 - Value of c is ", c )
c /= a
print ("Line 4 - Value of c is ", c )
c = 2
c %= a
print ("Line 5 - Value of c is ", c)
c **= a
print ("Line 6 - Value of c is ", c)
c //= a
print ("Line 7 - Value of c is ", c)
```

```
Line 1 - Value of c is  31
Line 2 - Value of c is  52
Line 3 - Value of c is  1092
Line 4 - Value of c is  52.0
Line 5 - Value of c is  2
Line 6 - Value of c is  2097152
Line 7 - Value of c is  99864
```

Python Membership Operators

Python's membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators as explained below

Operator	Description	Example
in	Evaluates to true, if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true, if it does not find a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

Example Program:

```
#Python Membership Operator
a = 10
b = 20
list = [1, 2, 3, 4, 5 ]
if ( a in list ):
    print ("Line 1 - a is available in the given list")
else:
    print ("Line 1 - a is not available in the given list")
if ( b not in list ):
    print ("Line 2 - b is not available in the given list")
else:
    print ("Line 2 - b is available in the given list")
c=b/a
if ( c in list ):
    print ("Line 3 - a is available in the given list")
else:
    print ("Line 3 - a is not available in the given list")
```

```
Line 1 - a is not available in the given list
Line 2 - b is not available in the given list
Line 3 - a is available in the given list
```

Python Identity Operators

Identity operators compare the memory locations of two objects. There are two Identity operators as explained below:

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here is results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here is not results in 1 if id(x) is not equal to id(y).

Example Program:

```
#Python Identity Operators
a = 20
b = 20
print ('Line 1','a=',a,':',id(a), 'b=',b,':',id(b))
if ( a is b ):
    print ("Line 2 - a and b have same identity")
else:
    print ("Line 2 - a and b do not have same identity")
if ( id(a) == id(b) ):
    print ("Line 3 - a and b have same identity")
else:
    print ("Line 3 - a and b do not have same identity")
b = 30
print ('Line 4','a=',a,':',id(a), 'b=',b,':',id(b))
if ( a is not b ):
    print ("Line 5 - a and b do not have same identity")
else:
    print ("Line 5 - a and b have same identity")
```

```
Line 1 a= 20 : 1956605600 b= 20 : 1956605600
Line 2 - a and b have same identity
Line 3 - a and b have same identity
Line 4 a= 20 : 1956605600 b= 30 : 1956605920
Line 5 - a and b do not have same identity
```

Operators Precedence

The following table lists all the operators from highest precedence to the lowest:

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Ccomplement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

Operator precedence affects the evaluation of an expression.
For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because the operator * has higher precedence than +, so it first multiplies 3*2 and then is added to 7.
Here, the operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom.

Example Program:

```
#Python Operator Precedence
a = 20
b = 10
c = 15
d = 5
print ("a:%d b:%d c:%d d:%d" % (a,b,c,d ))
e = (a + b) * c / d #( 30 * 15 ) / 5
print ("Value of (a + b) * c / d is ", e)
e = ((a + b) * c) / d # (30 * 15 ) / 5
print ("Value of ((a + b) * c) / d is ", e)
e = (a + b) * (c / d) # (30) * (15/5)
print ("Value of (a + b) * (c / d) is ", e)
e = a + (b * c) / d # 20 + (150/5)
print ("Value of a + (b * c) / d is ", e)
```

```
a:20 b:10 c:15 d:5
Value of (a + b) * c / d is  90.0
Value of ((a + b) * c) / d is  90.0
Value of (a + b) * (c / d) is  90.0
Value of a + (b * c) / d is  50.0
```

Python: Conditional statements, Control Structure, Functions

Objective:

To perform python programming using Conditional statements (If-Else, Nested if-else and if-elif-else ladder), Understanding of Control Structure: For Loop, While Loop), Concept of Functions (built-in, user defined functions, Recursion), Concept of Arrays, Structures.

Background and Procedure:

Conditional Statements

The 'if' Statement

The ability to control the flow of your program, letting it make decisions on what code to execute, is valuable to the programmer. The "if statement" allows you to control if a program enters a section of code or not based on whether a given condition is true or false.
Decision-making is the anticipation of conditions occurring during the execution of a program and specified actions taken according to the conditions.
Decision structures evaluate multiple expressions, which produce TRUE or FALSE as the outcome. You need to determine which action to take and which statements to execute if the outcome is TRUE or FALSE otherwise.
The if statement contains a logical expression using which the data is compared and a decision is made based on the result of the comparison,
If the Boolean expression evaluates to TRUE, then the block of statement(s) inside the if statement is executed. In Python, statements in a block are uniformly indented after the : symbol.
If Boolean expression evaluates to FALSE, then the first set of code after the end of block is executed.

Syntax:

if (CONDITION):
Execute the next statement

Program

```python
#Python if statement
check_value = 25
if (check_value == 25):
    if (check_value < 50):
        print (check_value, "is smaller than 20")
    if (check_value < 21):
        print (check_value, "is smaller than 21")
```

```
25 is smaller than 20
```

The 'else' Statement

An else statement can be combined with an if statement. An else statement contains a block of code that executes if the conditional expression in the if statement resolves to 0 or a FALSE value. The else statement is an optional statement and there could be at the most only one else statement following if.

Syntax:

if expression:
statement(s)
else:
statement(s)

Program

```python
#Python if-else statement
amount=int(input("Enter amount: "))
if amount<1000:
    discount=amount*0.05
    print ("Discount",discount)
else:
    discount=amount*0.10
    print ("Discount",discount)

print ("Net payable:",amount-discount)
```

```
Enter amount: 5666
Discount 566.6
Net payable: 5099.4
```

The 'if...elif...else' statements

The elif statement allows you to check multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE.

Like the else, the elif statement is optional. However, unlike else, for which there can be at the most one statement, there can be an arbitrary number of elif statements following an if.

- *Core Python does not provide switch or case statements as in other languages, but we can use if..elif...statements to simulate switch case as follows*

Syntax:

if expression1:
statement(s)
elif expression2:
statement(s)
elif expression3:
statement(s)
else:
statement(s)

Program

In the above example, discount is calculated on the input amount. Rate of discount is 5%, if the amount is less than 1000, and 10% if it is above 5000 and else takes care of all values by 15% rate of discount.

```
#Python if...elif...else  statements

amount=int(input("Enter amount: "))
if amount<1000:
    discount=amount*0.05
    print ("Discount",discount)
elif amount<5000:
    discount=amount*0.10
    print ("Discount",discount)
else:
    discount=amount*0.15
    print ("Discount",discount)
print ("Net payable:",amount-discount)
```

```
Enter amount: 590
Discount 29.5
Net payable: 560.5
```

The if-else Statements in One Line (Ternary Operator)

Syntax:

if expression: set of statement to execute if condition is true
else: statements to execute if condition is false

OR
a if condition else b

Program

```
#Pythom ternary operator
# a if condition else b
i= 10

print("condition true here") if i>3 else print("condition false here")
```

```
condition true here
```

The Nested if-else Statements

if statement can also be checked inside other if statement. This conditional statement is called a nested if statement. This means that inner if condition will be checked only if outer if condition is true and by this, we can see multiple conditions to be satisfied.

Syntax

```
if (condition1):
  # Executes when condition1 is true
  if (condition2):
    # Executes when condition2 is true
  # if Block is end here
# if Block is end here
```

Program

```
#Python Nested if-else statment
value = 55
if value > 10:
    print("The value is Above ten")
    if value > 20:
        print("and also above 20..")
    if value > 40:
        print("even also above 40!")
    else:
        print("but not above 55.")
```

```
The value is Above ten
and also above 20..
even also above 40!
```

The 'if-elif-else' Ladder

There may be a situation when you want to check for another condition after a condition resolves to true. In such a situation, you can use the nested if construct. In a nested if construct, you can have an if-.elif-else construct inside another If-elif-else construct.

Syntax

if expression1:
statement(s)
if expression2:
statement(s)
elif expression3:
statement(s)
else
statement(s)
elif expression4:
statement(s)
else:
statement(s)

Program

```
#Python if-elif-else Ladder
price = 50
quantity = 5
amount = price*quantity

if amount > 100:
    if amount > 500:
        print("Amount is greater than 500")
    else:
        if amount < 500 and amount > 400:
            print("Amount is between 400 and 500")
        elif amount < 500 and amount > 300:
            print("Amount is between 300 and 500")
        else:
            print("Amount is between 200 and 500")
elif amount == 100:
    print("Amount is 100")
else:
    print("Amount is less than 100")
```

```
Amount is between 200 and 500
```

Understanding of Control Structure

Loop

In general, statements are executed sequentially: The first statement is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times. A loop statement allows us to execute a statement or group of statements multiple times. The following diagram illustrates a loop statement.

For Loop

In Python, for keyword provides a more comprehensive mechanism to constitute a loop. The **for** loop is used with sequence types such as list, tuple, set, range, etc. The body of the for loop is executed for each member element in the sequence.

Syntax

for x in sequence:
  statement1
  statement2
  ...
  statementN

```
#Python for loop

str1 = 'programming'

for i in str1:
    print(i)
```

```
p
r
o
g
r
a
m
m
i
n
g
```

```python
#Python for loop - list
Colors = ["red", "yellow", "blue"]
for x in Colors:
    print(x)
```

```
red
yellow
blue
```

```python
for x in range(3):
    print("Printing:", x)
```

```
Printing: 0
Printing: 1
Printing: 2
```

While Loop

In most computer programming languages, a while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.

The 'for' loop does something a fixed number of times. But in a case where you don't know how many times you want to do something before you start the loop, you use a different loop: the while loop. Here, statement(s) may be a single statement or a block of statements. The condition may be any expression, and true is any non-zero value. The loop iterates while the condition is true. When the condition becomes false, program control passes to the line immediately following the loop.

In Python, all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements

Syntax

while expression:
  statement(s)

```python
#Python While loop
count = 0
while (count < 9):
    print("The count is:", count)
    count = count + 1

print ("End!")
```

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
End!
```

```python
#Python While loop
# numbers up to
# sum = 1+2+3+...+n

# To take input from the user,
n = int(input("Enter n: "))

# initialize sum and counter
sum = 0
i = 1

while i <= n:
    sum = sum + i
    i = i+1    # update counter

# print the sum
print("The sum is", sum)
```

```
Enter n: 5
The sum is 15
```

```python
#Python While loop - with else

i = 0
while i < 4:
    i += 1
    print(i)
    #break
else:  # Executed because no break in while
    print("No Break\n")
```

```
1
2
3
4
No Break
```

Concept of Functions

A function can be called as a section of a program that is written once and can be executed whenever required in the program, thus making code reusability.

- Keyword def that marks the start of the function header.
- A function name to uniquely identify the function. Function naming follows the same rules of writing identifiers in Python.
- Parameters (arguments) through which we pass values to a function. They are optional.
- A colon (:) to mark the end of the function header.
- Optional documentation string (docstring) to describe what the function does.
- One or more valid python statements that make up the function body. Statements must have the same indentation level (usually 4 spaces).
- An optional return statement to return a value from the function.
- Once we have defined a function, we can call it from another function, program, or even the Python prompt. To call a function we simply type the function name with appropriate parameters.

Syntax

def function_name(argument1, argument2, ...) :
    statement_1
    statement_2
    ....

Scope and Lifetime of function variables:

- Scope of a variable is the portion of a program where the variable is recognized. Parameters and variables defined inside a function are not visible from outside the function. Hence, they have a local scope.
- The lifetime of a variable is the period throughout which the variable exists in the memory. The lifetime of variables inside a function is as long as the function executes.
- They are destroyed once we return from the function. Hence, a function does not remember the value of a variable from its previous calls.

Python User-defined Functions

```
#Python Function
#user-defined functions

def Summation(value1,value2):
    sum = value1 + value2
    return sum

num1 = 5
num2 = 6

print("The sum is", Summation(num1, num2))
```

```
The sum is 11
```

```
# Program make a simple calculator that can add, subtract, multiply and divide using functions

# This function adds two numbers
def add(x, y):
    return x + y

# This function subtracts two numbers
def subtract(x, y):
    return x - y

# This function multiplies two numbers
def multiply(x, y):
    return x * y

# This function divides two numbers
def divide(x, y):
    return x / y

print("Select operation.")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")

# Take input from the user
choice = input("Enter choice(1/2/3/4):")

num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))

if choice == '1':
    print(num1,"+",num2,"=", add(num1,num2))
elif choice == '2':
    print(num1,"-",num2,"=", subtract(num1,num2))
elif choice == '3':
    print(num1,"*",num2,"=", multiply(num1,num2))
elif choice == '4':
    print(num1,"/",num2,"=", divide(num1,num2))
else:
    print("Invalid input")
```

```
Select operation.
1.Add
2.Subtract
3.Multiply
4.Divide
Enter choice(1/2/3/4):2
Enter first number: 23
Enter second number: 66
23 - 66 = -43
```

Python Built-in Functions

The abs() method takes a single argument:

```
#Python Function
#built-in functions

integer = -20
print('Absolute value of -20 is:', abs(integer))

floating = -30.33
print('Absolute value of -30.33 is:', abs(floating))
```

```
Absolute value of -20 is: 20
Absolute value of -30.33 is: 30.33
```

Concept of Arrays

An array is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together. This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array (generally denoted by the name of the array)

Array in Python can be created by importing array module. array(data_type, value_list) is used to create an array with data type and value list specified in its arguments.

```python
#Python Arrays

# importing "array" for array creations
import array as arr

# creating an array with integer type
a = arr.array('i', [1, 2, 3])

# printing  array
print ("The new created integer type array is : ",  end =" ")
for i in range (0, 3):
    print (a[i], end =" ")
print()

# creating an array with float type
b = arr.array('d', [2.5, 3.2, 3.3])

# printing array
print ("\n The new created float type array is : ",end =" ")
for i in range (0, 3):
    print (b[i], end =" ")

# creating an array with char type
b = arr.array('u', ['a','e','i','o','u'])

# printing array
print ("\n The new created char type array is : ", end =" ")
for i in range (0, 5):
    print (b[i], end =" ")
```

```
The new created integer type array is :  1 2 3

 The new created float type array is :  2.5 3.2 3.3
 The new created char type array is :  a e i o u
```

```python
# Adding Elements to a Array

# importing "array" for array creations
import array as arr

# array with int type
a = arr.array('i', [1, 2, 3])

print ("Array before insertion : ", end =" ")
for i in range (0, 3):
    print (a[i], end =" ")
print()

# inserting array using
# insert() function
a.insert(1, 4)

print ("Array after insertion : ", end =" ")
for i in (a):
    print (i, end =" ")
print()

# array with float type
b = arr.array('d', [2.5, 3.2, 3.3])

print ("Array before insertion : ", end =" ")
for i in range (0, 3):
    print (b[i], end =" ")
print()

# adding an element using append()
b.append(4.4)

print ("Array after insertion : ", end =" ")
for i in (b):
    print (i, end =" ")
print()
```

```
Array before insertion :  1 2 3
Array after insertion :  1 4 2 3
Array before insertion :  2.5 3.2 3.3
Array after insertion :  2.5 3.2 3.3 4.4
```

```python
# accessing of element from list

# importing array module
import array as arr

# array with int type
a = arr.array('i', [1, 2, 3, 4, 5, 6])

# accessing element of array
print("Access element is: ", a[0])

# accessing element of array
print("Access element is: ", a[3])

# array with float type
b = arr.array('d', [2.5, 3.2, 3.3])

# accessing element of array
print("Access element is: ", b[1])

# accessing element of array
print("Access element is: ", b[2])
```

```
Access element is:  1
Access element is:  4
Access element is:  3.2
Access element is:  3.3
```

```
# Removal of elements in a Array

# importing "array" for array operations
import array

# initializing array with array values
# initializes array with signed integers
arr = array.array('i', [1, 2, 3, 1, 5])

# printing original array
print ("The new created array is : ", end ="")
for i in range (0, 5):
    print (arr[i], end =" ")
print ("\r")

# using pop() to remove element at 2nd position
print ("The popped element is : ", end ="")
print (arr.pop(2))

# printing array after popping
print ("The array after popping is : ", end ="")
for i in range (0, 4):
    print (arr[i], end =" ")

print("\r")
```

```
# using remove() to remove 1st occurrence of 1
arr.remove(1)

# printing array after removing
print ("The array after removing is : ", end ="")
for i in range (0, 3):
    print (arr[i], end =" ")
```

```
The new created array is : 1 2 3 1 5
The popped element is : 3
The array after popping is : 1 2 1 5
The array after removing is : 2 1 5
```

Python Packages, Plotting and Single Layer Perceptron (SLP)

Objective:

To perform the basic machine learning artificial neural network (Single Layer Perceptron) algorithm and determine the outcome with familiarization with python packages Pandas, Numpy, Sci-kit learn, and Mat-plot library.

Background and Procedure:

Python Packages and Plotting
With Python you don't get so much out of the box. Instead of having all of its functionality built into its core, you need to install different packages for different topics.

Python Packages for Science and Numerical Computations:
Some important Python Packages for Science and Numerical Computations are:

NumPy:
NumPy is the fundamental package for scientific computing with Python
SciPy:
SciPy is a free and open-source Python library used for scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering.
Matplotlib:
Matplotlib is a Python 2D plotting library.
Pandas:
Pandas Python Data Analysis Library

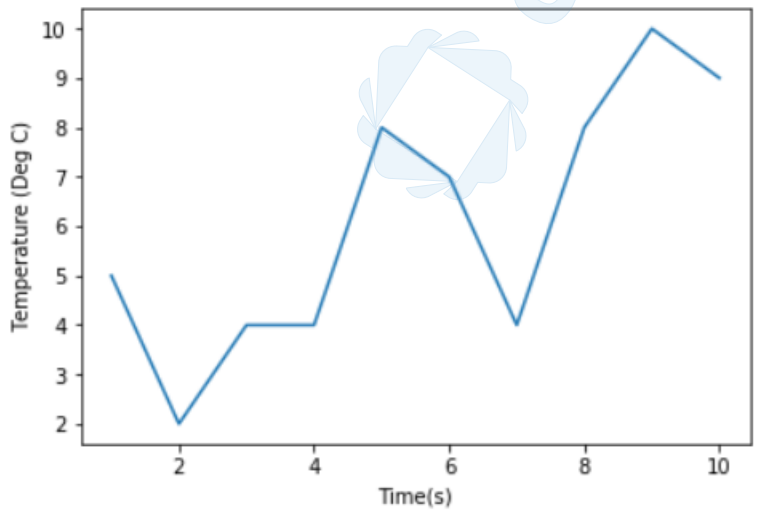
Built-in Functions:
Python consists of lots of built-in functions. Some examples are the print (function that we already have used (perhaps without noticing it is actually a Built-in function).
Python also consists of different Modules, Libraries or Packages. These Modules, Libraries or Packages consists of lots of predefined functions for different topics or areas, such as mathematics, plotting, handling database systems, etc.
More functions we will explore later

Python Standard Library:

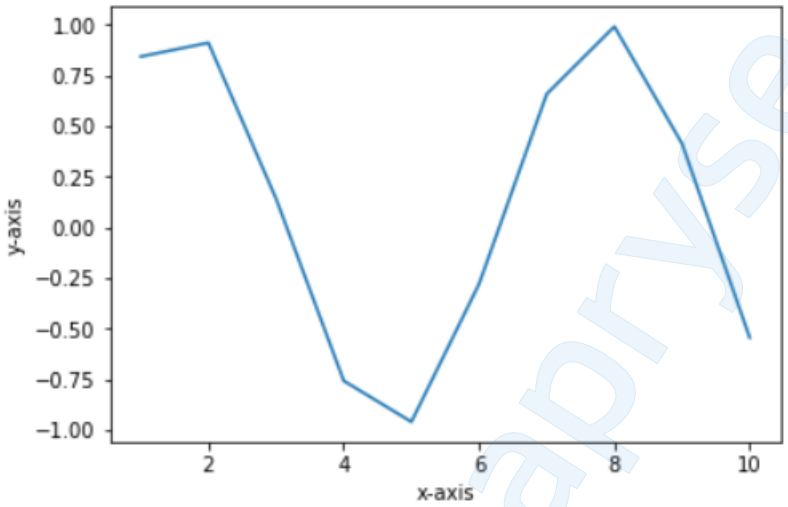
Python allows you to split your program into modules that can be reused in other Python programs. It comes with a large collection of standard modules that you can use as the basis of your programs.
The Python Standard Library consists of different modules for handling file I/O, basic mathematics, etc. You don't need to install these separately, but you need to important them when you want to use some of these modules or some of the functions within these modules.
The math module has all the basic math functions you need, such as: Trigonometric functions: sin(x), cos(x), etc. Logarithmic functions: log(), log10(), etc.
Constants like pi, e, inf, nan, etc.

Plotting in Python:
Typically you need to create some plots or charts. In order to make plots or charts in Python you will need an external library. The most used library is Matplotlib.
Matplotlib is a Python 2D plotting library.

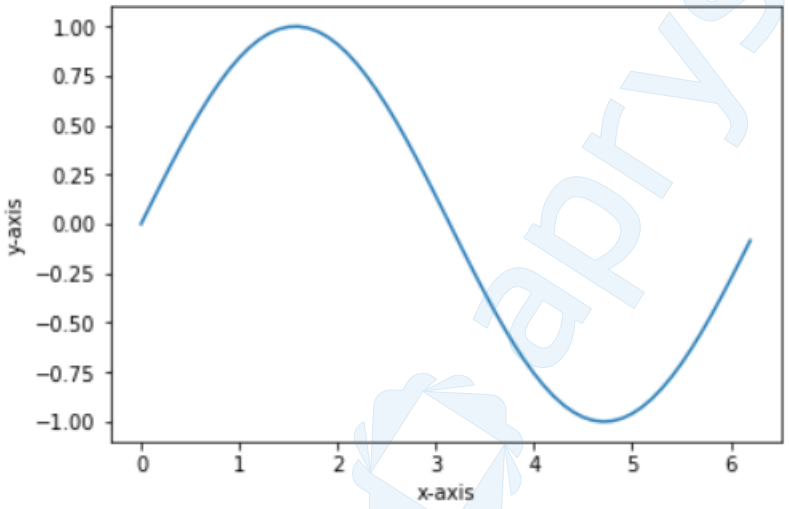
```
import matplotlib.pyplot as plt
x = [1,2,3,4,5,6,7,8,9,10]
y = [5,2,4,4,8,7,4,8,10,9]

plt.plot(x,y)
plt.xlabel('Time(s)')
plt.ylabel('Temperature (Deg C)')
plt.show()
```



```
import numpy as np
import matplotlib.pyplot as plt
x = [1,2,3,4,5,6,7,8,9,10]
y = np.sin(x)

plt.plot(x,y)
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.show()
```

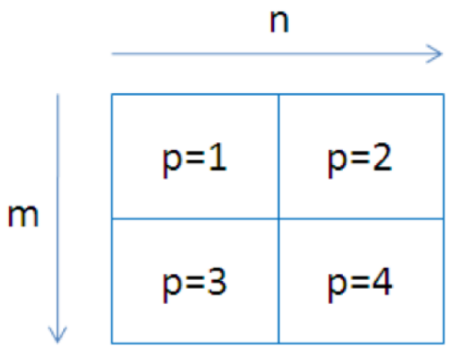


```
import numpy as np
import matplotlib.pyplot as plt

xstart = 0
xstop = 2*np.pi
increment = 0.1

x = np.arange(xstart,xstop,increment)
y = np.sin(x)

plt.plot(x,y)
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.show()
```

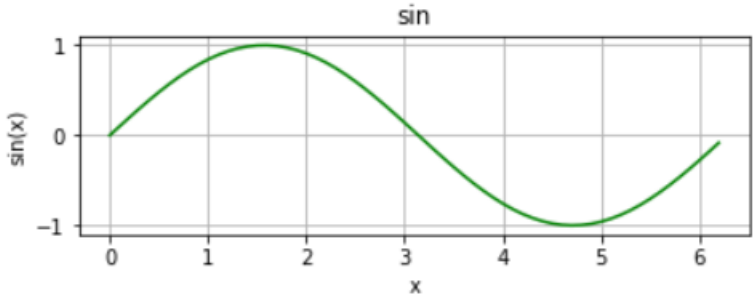


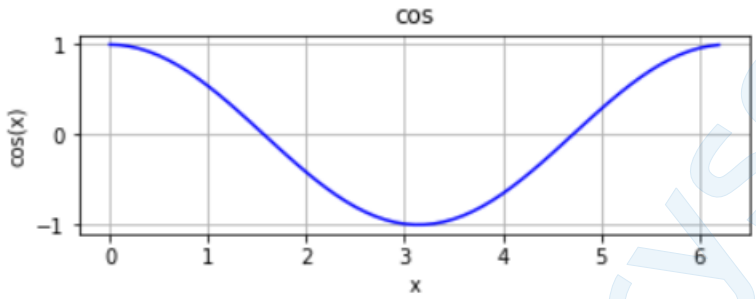
Subplots:

The subplot command enables you to display multiple plots in the same window. Typing "subplot(m,n,p)" partitions the figure window into an m-by-n matrix of small subplots and selects the subplot for the current plot. The plots are numbered along the first row of the figure window, then the second row, and soon.

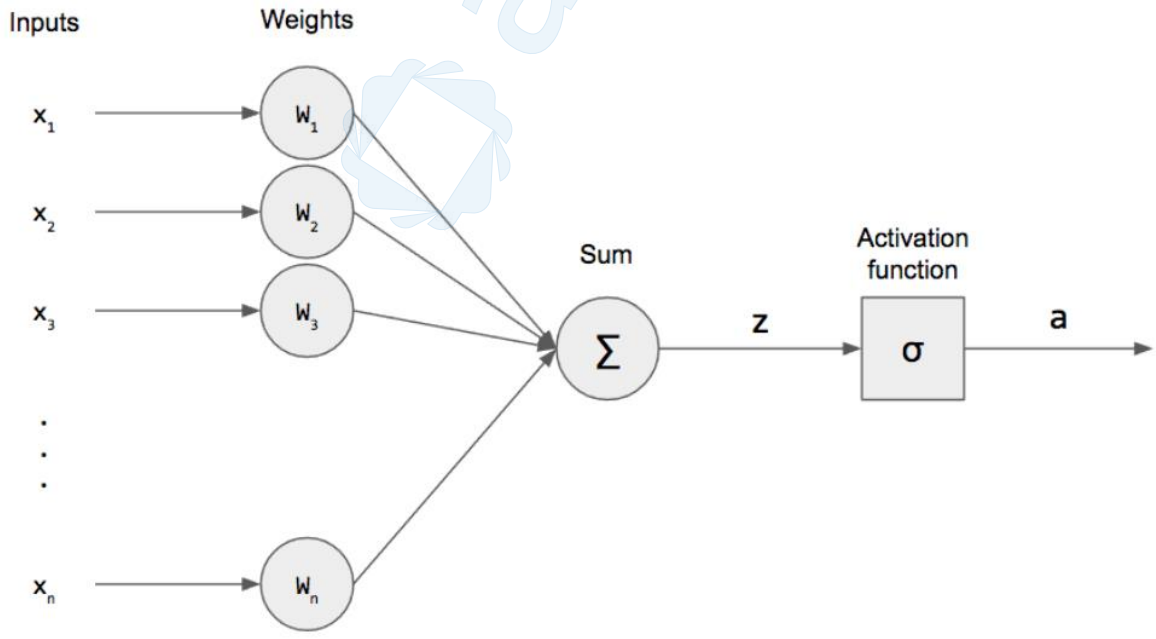



```
# Subplots
# The subplot command enables you to display multiple plots in the same window.
# Typing "subplot(m,n,p)" partitions the figure window into an
# m-by-n matrix of small subplots and selects the subplot for the current plot.
# The plots are numbered along the first row of the figure window,
#then the second row, and soon.

import numpy as np
import matplotlib.pyplot as plt

xstart = 0
xstop = 2*np.pi
increment = 0.1

x = np.arange(xstart,xstop,increment)
y = np.sin(x)
z = np.cos(x)

plt.subplot(2,1,1)
plt.plot(x,y,'g')
plt.title('sin')
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.grid()
plt.show()

plt.subplot(2,1,2)
plt.plot(x,z,'b')
plt.title('cos')
plt.xlabel('x')
plt.ylabel('cos(x)')
plt.grid()
plt.show()
```





Single-Layer Perceptron



Single Layer Perceptron Mathematical Model

In this model, we have n binary **inputs** (usually given as a vector) and exactly the same number of **weights** $W_1, ..., W_n$. We multiply these together and sum them up. We denote this as z and call it the **pre-activation**.

$$z = \sum_{i=1}^{n} W_i x_i = W^T x$$

There is another term, called the **bias**, that is just a constant factor.

$$z = \sum_{i=1}^{n} W_i x_i + b = W^T x + b$$

For mathematical convenience, we can actually incorporate it into our weight vector as W_0 and set $x_0 = +1$ for all of our inputs.

$$z = \sum_{i=0}^{n} W_i x_i = W^T x$$

After taking the weighted sum, we apply an activation function, σ, to this and produce an activation a. The activation function for perceptrons is sometimes called a **step function** because, if we were to plot it, it would look like a stair.

$$\sigma(q) = \begin{cases} 1 & q \geq 0 \\ 0 & q < 0 \end{cases}$$

In other words, if the input is greater than or equal to 0, then we produce an output of 1. Otherwise, we produce an output of 0.

$$a = \sigma(W^T x)$$

Single Layer Perceptron Learning Algorithm

For perceptron's learning, we change the weight vector (and bias). The weight vector is a parameter to the perceptron. We need to keep changing it until we can correctly classify each of our inputs. For that, we need to write an update rule for our weight vector so that we can appropriately change it:

$$w \leftarrow w + \Delta w$$

First, we can define the error as the difference between the desired output d and the predicted output y.

$$e = d - y$$

Notice that when d and y are same (both are 0 or both are 1), we get 0. When they are different, (0 and 1 or 1 and 0), we can get either 1 or -1. This directly corresponds to exciting and inhibiting our perceptron. We multiply this with the input to tell our perceptron to change our weight vector in proportion to our input.

$$w \leftarrow w + \eta \cdot e \cdot x$$

There is a hyperparameter η that is called the learning rate. It is just a scaling factor that determines how large the weight vector updates should be. When the error is 0, i.e., the output is

what we expect, then we don't change the weight vector at all. When the error is nonzero, we update the weight vector accordingly.

Example

Suppose that we are going to work on AND Gate problem. The gate returns if and only if both inputs are true.

X_1	X_2	Y
0	0	0
0	1	0
1	0	0
1	1	1

We are going to set weights randomly. Let's say that w1 = 0.9 and w2 = 0.9

a) Round 1

We will apply 1st instance to the perceptron. x1 = 0 and x2 = 0.

Sum unit will be 0 as calculated below

$\Sigma = x1 * w1 + x2 * w2 = 0 * 0.9 + 0 * 0.9 = 0$

Activation unit checks sum unit is greater than a threshold. If this rule is satisfied, then it is fired and the unit will return 1, otherwise it will return 0. BTW, modern neural networks architectures do not use this kind of a step function as activation.

Activation threshold would be 0.5.

Sum unit was 0 for the 1st instance. So, activation unit would return 0 because it is less than 0.5. Similarly, its output should be 0 as well. We will not update weights because there is no error in this case.

Let's focus on the 2nd instance. x1 = 0 and x2 = 1.

Sum unit: Σ = x1 * w1 + x2 * w2 = 0 * 0.9 + 1 * 0.9 = 0.9

Errors:

Activation unit will return 1 because sum unit is greater than 0.5. However, output of this instance should be 0. This instance is not predicted correctly. That's why, we will update weights based on the error.

ε = actual – prediction = 0 – 1 = -1

We will add error times learning rate value to the weights. Learning rate would be 0.5. BTW, we mostly set learning rate value between 0 and 1.

w1 = w1 + α * ε = 0.9 + 0.5 * (-1) = 0.9 – 0.5 = 0.4

w2 = w2 + α * ε = 0.9 + 0.5 * (-1) = 0.9 – 0.5 = 0.4

Focus on the 3rd instance. x1 = 1 and x2 = 0.

Sum unit: Σ = x1 * w1 + x2 * w2 = 1 * 0.4 + 0 * 0.4 = 0.4

Activation unit will return 0 this time because output of the sum unit is 0.5 and it is less than 0.5. We will not update weights.

Mention the 4rd instance. x1 = 1 and x2 = 1.

Sum unit: Σ = x1 * w1 + x2 * w2 = 1 * 0.4 + 1 * 0.4 = 0.8

Activation unit will return 1 because output of the sum unit is 0.8 and it is greater than the threshold value 0.5. Its actual value should 1 as well. This means that 4th instance is predicted correctly. We will not update anything.

b) Round 2

In previous round, we've used previous weight values for the 1st instance and it was classified correctly. Let's apply feed forward for the new weight values.

Remember the 1st instance. x1 = 0 and x2 = 0.

Sum unit: Σ = x1 * w1 + x2 * w2 = 0 * 0.4 + 0 * 0.4 = 0.4

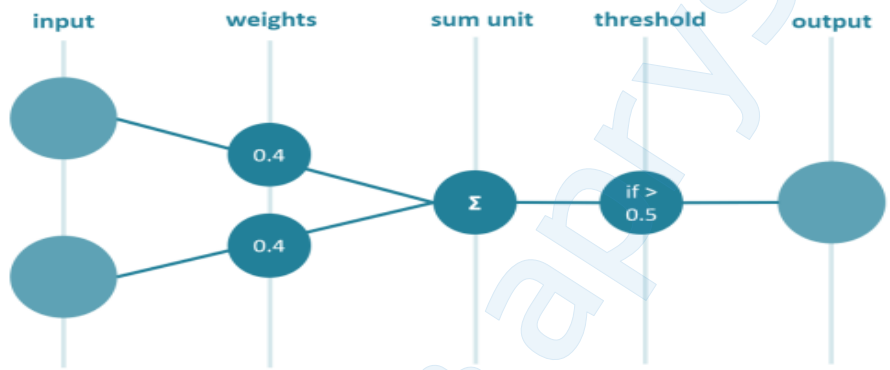
Activation unit will return 0 because sum unit is 0.4 and it is less than the threshold value 0.5. The output of the 1st instance should be 0 as well. This means that the instance is classified correctly. We will not update weights.

Feed forward for the 2nd instance. x1 = 0 and x2 = 1.

Sum unit: $\Sigma = x1 * w1 + x2 * w2 = 0 * 0.4 + 1 * 0.4 = 0.4$

Activation unit will return 0 because sum unit is less than the threshold 0.5. Its output should be 0 as well. This means that it is classified correctly and we will not update weights.

We've applied feed forward calculation for 3rd and 4th instances already for the current weight values in the previous round. They were classified correctly.



Learning Term:

We should continue this procedure until learning completed. We can terminate the learning procedure here. Luckily, we can find the best weights in 2 rounds.

Updating weights means learning in the perceptron. We set weights to 0.9 initially but it causes some errors. Then, we update the weight values to 0.4. In this way, we can predict all instances correctly.

Experiment

Implementation of Single Layer Perceptron

We can create perceptron that act like gates: they take 2 binary inputs and produce a single binary output.

1. Defining inputs size, the learning rate and number of epochs as inputs and weights vector for the network. One is added to the input size to include the bias in the weight vector.
2. Implementing activation function which returns 1 if the input is greater than or equal to 0 and 0 otherwise.
3. Creating prediction function to run an input through the perceptron and return an output. The bias is added into the input vector then we can simply compute the inner product and apply the activation function.
4. Perceptron Learning Algorithm Code: Create a function to keep applying this update rule until our perceptron can correctly classify all of our inputs. We need to keep iterating through our training data until this happens; one epoch is when our perceptron has seen all of the training data once. Usually, we run our learning algorithm for multiple epochs. We can create a function, given inputs and desired outputs, run our perceptron learning algorithm. We keep updating the weights for a number of epochs, and iterate through the entire training set. We insert the bias into the input when performing the weight update. Then we can create our prediction, compute our error, and perform our update rule.

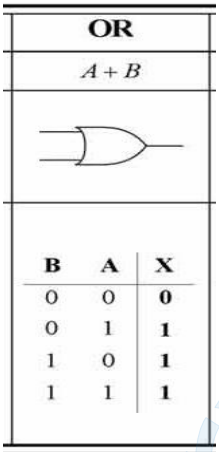
```
#Implement single layer perceptron network
import numpy as np
class SLP(object):
    #"__init__" is a reseved method in python classes.
    #It is known as a constructor in object oriented concepts.
    #This method called when an object is created from the class and it allow the class to initialize the attributes of a class.
    def __init__(self, input_size, learning_rate=1, epochs=1000):
        #zeros() function is used to get a new array of given shape and type, filled with zeros.
        self.Weights = np.zeros(input_size+1)
        # add one for bias
        print("Creating weight vector, one-dimensional array woth zeros", self.Weights) #print one-dimensional array with zeros
        self.epochs = epochs
        self.learning_rate = learning_rate

    def activation_function(self, input_value):
        #returns output 1 if the input is greater than or equal to 0 and 0 otherwise as output.
        #return (input_value >= 0).astype(np.float32)
        return 1 if input_value >= 0 else 0

    def predict(self, input_value):
        #pre-activattion/bias/weighted sum
        #dot() performs both element multiplication and sum
        #T is array property returning the array transposed.
        z = self.Weights.T.dot(input_value)
        a = self.activation_function(z)
        return a
```

```
    def perceptronLearning(self, given_input, desired_output):
        for j in range(self.epochs):
            for i in range(desired_output.shape[0]): #Shape[0] is n.shape is a tuple that always gives dimensions of the array.
                x = np.insert(given_input[i], 0, 1)
                y = self.predict(x) #create our prediction as predicted output
                e = desired_output[i] - y #compute error
                print("Error:", e)
                print("Predicted output:", y)
                #print("Predicted output:", desired_output)
                self.Weights = self.Weights + self.learning_rate * e * x #perform weight update rule


#AND Gate
given_input = np.array([
        [0, 0],
        [0, 1],
        [1, 0],
        [1, 1]
    ])
desired_output = np.array([0, 0, 0, 1])
print("Given Input \n", given_input)
print("Desired Output",desired_output)
slp = SLP(input_size=2)
slp.perceptronLearning(given_input, desired_output)
print("Input Weights with bias", slp.Weights)
print("Learning rate:", slp.learning_rate)
print("Total epochs:", slp.epochs)
```

Basic perceptron can generalize any kind of linear problem. The both AND and OR Gate problems are linearly separable problems. On the other hand, this form cannot generalize non-linear problems such as XOR Gate. Perceptron evolved to multilayer perceptron to solve non-linear problems and deep neural networks were born.

Lab Exercises:

1. To reinforce the perceptron, apply learning procedure for OR Gate. The gate returns 0 if and only if both inputs are 0. Do not hesitate to change the initial weights and learning rate values.



B	A	X
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

2. Can this form generalized the same algorithm for nonlinear problem XOR gate , why or why not?

_____
_____
_____
_____
_____

Conclusion:

_____
_____
_____
_____
_____
_____

Artificial Neural Network (Multi-Layer Perceptron): Biological Binary Model

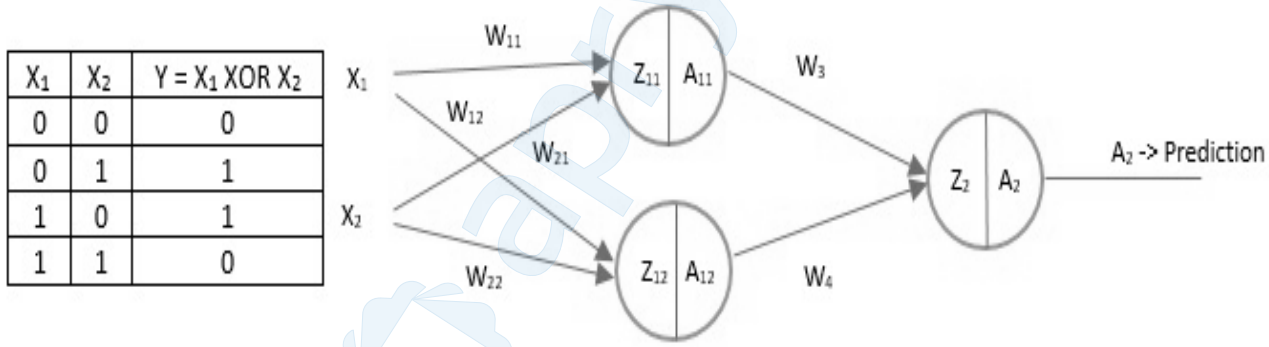
Objective:

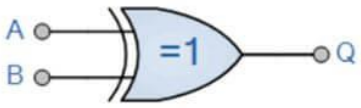
To perform the basic machine learning artificial neural network (Multi Layer Perceptron) algorithm and determine the outcome with familiarization with python packages Pandas, Numpy, Sci-kit learn, and Mat-plot library.

Background and Procedure:

Multi Layer Perceptron Mathematical Model

X_1	X_2	$Y = X_1$ XOR X_2
0	0	0
0	1	1
1	0	1
1	1	0



Symbol	Truth Table		
 2-input Ex-OR Gate	B	A	Q
	0	0	0
	0	1	1
	1	0	1
	1	1	0
Boolean Expression $Q = A \oplus B$	A **OR** B but NOT **BOTH** gives Q		

Multi-Layer Perceptron Mathematical Model and Algorithm

For simplicity, we consider weights 'W_{11}','W_{12}','W_{21}' and 'W_{22}' as weight vector 'W_1'. Weights 'W_3' and 'W_4' as weight vector 'W_2'.

Here 'Z' is the dot product of weight vector 'W' and input vector 'X' added with bias. Again, we vectorize 'Z_{11}' and 'Z_{12}' as 'Z_1' and 'Z_2' remains the same.

Lastly, 'A' is the activation function. We used sigmoid activation functions in our network. Vectorizing 'A_{11}' and 'A_{12}' gives us 'A_1' and 'A_2' is kept the same.

We consider Z For simplicity, we didn't include bias in the network. A model can work without bias unit but it cannot work without weights, as then machine will not learn anything if weights don't change.

After all calculations, 'Y' is our network prediction.

Lets see what happens inside a neural network training process. It has 4 parts:

a) Forward Propagation:

In forward propagation, all 'Z' and 'A' will be calculated until we get to the end of our network giving us our prediction 'Y'. This process is quite straight-forward.

Mathematically Z and A are given as,

$$Z_1 = W_1.X+b_1 \quad \text{---------- (1)}$$

$$A_1 = \frac{1}{1+e^{-Z_1}} \quad \text{---------- (2)}$$

$$Z_2 = W_2.A_1+b_2 \quad \text{---------- (3)}$$

$$Y = A_2 = \frac{1}{1+e^{-Z_2}} \quad \text{---------- (4)}$$

Where 'Z_1' is dot product of weight and input vector (we didn't include bias b_1 in our network), 'A_1' is the mathematical formula of sigmoid function, 'Z_2' is the dot product of hidden layer output 'A_1' and 'W_2' (ignoring bias b_2) and 'Y' is the output prediction given by sigmoid activation in output neuron.

b) Calculating Loss function value:

Loss function value is an estimate of how our network is performing. If this value is high and keeps increasing, that means our network is not performing well. If this value is decreasing and at the end of iterations, reaches a low value, then we can say that our network is doing well, our model is learning

because loss functions is converging. For binary outputs, we use formula of binary cross-entropy for loss function value in Equation (5).

$$L = -\left(\frac{1}{m}\right) * \sum(Y * log(\mathrm{A2}) + (1 - \mathrm{Y}) * \log(1 - \mathrm{A2})) \quad \text{---------- (5)}$$

Where 'L' is the loss value,
'Y' is the output that we already know, since this is supervised learning, we will train our model with known outputs contained in 'Y',
'A_2' is output of our network after each forward propagation.
Using this formula, we plot the loss function value to see if our network loss is decreasing or not after iterations.

c) Backward Propagation:

In backward propagation, we back propagate the loss that we encountered in our model to tweak the weights and biases such that it decreases the loss function value. Considering our network, we start backpropagation with A2 and we try to find the rate of change of loss with respect to weight vector W_2 using chain rule of derivative and then use gradient descent's formula to tweak the weights. Equation 6 shows that process:

$$\frac{\partial L}{\partial W2} = \frac{\partial L}{\partial A2} * \frac{\partial A2}{\partial Z2} * \frac{\partial Z2}{\partial W2} \quad \text{---------- (6)}$$

We use the above equation to find rate of change of loss with respect to weight vector 'W_2' where $\frac{\partial L}{\partial A2}$ is found by taking derivative of loss function 'L' with respect to activation function 'A_2'. After derivative, we get output shown in Equation 7,

$$\frac{\partial L}{\partial A2} = (-\frac{1}{m})(\frac{Y-A2}{A2(1-A2)}) \quad \text{---------- (7)}$$

Then we find $\frac{\partial A2}{\partial Z2}$ by taking derivative of 'A_2' with respect to 'Z_2'. Taking derivative of sigmoid function in equation 4 gives us the following output:

$$\frac{\partial A2}{\partial Z2} = \mathrm{A2}*(1\text{-}\mathrm{A2}) \quad \text{---------- (8)}$$

Lastly, we take derivative of 'Z_2' with respect to 'W_2' in equation 3 as,

$$\frac{\partial Z2}{\partial W2} = A1 \quad \text{---------- (9)}$$

Now we try to find the rate of change of Loss with respect to Weight vector 'W1'. First, we find $\frac{\partial A1}{\partial Z1}$

$$\frac{\partial A1}{\partial Z1} = \mathrm{A_1} * (1\text{-}\mathrm{A_1}) \quad \text{---------- (10)}$$

Then we find $\frac{\partial Z1}{\partial W1}$ by taking derivative of equation 1,

$$\frac{\partial Z1}{\partial W1} = X \quad \text{---------- (11)}$$

Lastly, we multiply $\frac{\partial Z1}{\partial W1}$ and $\frac{\partial A1}{\partial Z1}$ with the input derivative coming from output layer i.e. W_2 and $\frac{\partial L}{\partial Z2}$, we get

$$\frac{\partial L}{\partial W1} = X * A1 * (1 - A1) * W2 * (A2 - Y) \quad \text{---------- (12)}$$

d) Weight update via Gradient Descent

Finally, we update the weights using gradient descent formula as,

$$W = W - lr * \frac{\partial L}{\partial W} \quad \text{---------- (13)}$$

Where 'W' is weight vector, 'lr' is learning rate and $\frac{\partial L}{\partial W}$ is the rate of change of Loss w.r.t. weight. Thus, weights are updated per iteration also called EPOCH which is made up of one forward and one backward propagation, and we keep doing these EPOCHS and see if loss is decreasing until we settle at a point where loss is minimum and our classes are predicted perfectly.

Multi-Layer Perceptron Code (XOR Gate)

Introduction:

In this implementation of Multi-Layer Perceptron neural network, a single hidden layer neural network is used, with sigmoid activation function in hidden layer units and sigmoid activation function for output layer too, since the output of XOR logic is binary i.e. 0 or 1 only one neuron is in the output layer. This time we are not adding bias with inputs as SLP. However, the algorithm will run fine without adding bias but can't without weight updates. The mathematics and basic logic behind the neural network is also explained.

[Hints]

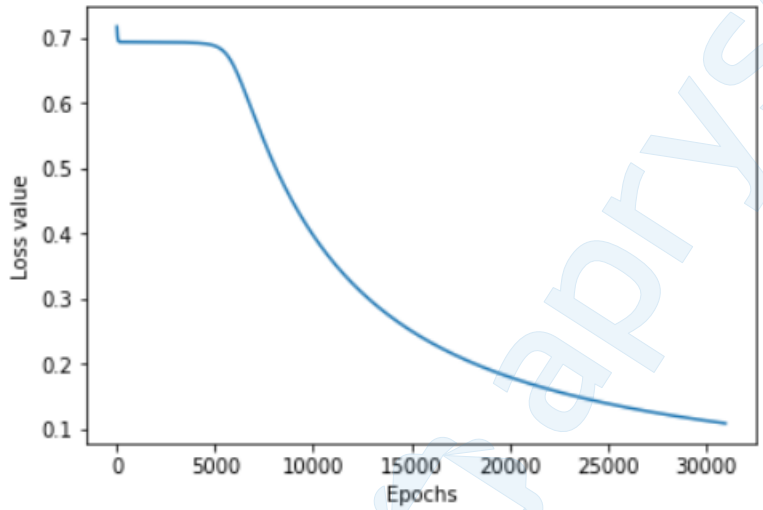
X = inputs
Y= desired output
No_x = number of inputs
No_y = number of neurons in output layer
No_h = Number of neurons in hidden layer
Tot = Total training examples
Lr = learning rate
W1, w2 = hidden layers weights
Losses = error or loss calculation
frwd_prop = defining forward propagation method
sigmoid = defining sigmoid method
z = pre-activation /weights vector sum
z1,a1,z2,a2 = weight updates and activation variables (forward propagation)
dz2,dw2,dz1,dw1 = derivative of weights and weights updates (w.r.t. weight vector and activation function)
mlp_test = testing multilayer perceptron learning and network prediction

Rewrite the Coding as Define Below and Place Output Screen shot:

```
#Implementing MLP
#Defining inputs and structure of neural network
import numpy as np
import matplotlib.pyplot as plt

x=np.array([[0,0,1,1],[0,1,0,1]])
y=np.array([[0,1,1,0]])
no_x = 2
no_y = 1
no_h = 2
tot = x.shape[1]
lr = 0.1
np.random.seed(2) # Define random seed for consistent results
w1 = np.random.rand(no_h,no_x)
w2 = np.random.rand(no_y,no_h)
losses = []
```

```
#Defining sigmoid activation function, forward and backward propagation with loss calculation for hidden layer and output
#sigmoid activation function for hidden layer and output
def sigmoid(z):
    z= 1/(1+np.exp(-z))
    return z
# Forward propagation
def frwd_prop(w1,w2,x):
    z1 = np.dot(w1,x)
    a1 = sigmoid(z1)
    z2 = np.dot(w2,a1)
    a2 = sigmoid(z2)
    return z1,a1,z2,a2

#Calculating Loss value
epochs = 20000
for i in range(epochs):
    z1,a1,z2,a2 = frwd_prop(w1,w2,x)
    loss = -(1/tot)*np.sum(y*np.log(a2)+(1-y)*np.log(1-a2))
    losses.append(loss)
    da2,dw2,dz1,dw1 = back_prop(tot,w1,w2,z1,a1,z2,a2,y)
    w2 = w2-lr*dw2
    w1 = w1-lr*dw1

# plotting losses to see how network is performing
plt.plot(losses)
plt.xlabel("Epochs")
plt.ylabel("Loss value")
```

```
# Backward propagation
def back_prop(tot,w1,w2,z1,a1,z2,a2,y):

    dz2 = a2-y
    dw2 = np.dot(dz2,a1.T)/tot
    dz1 = np.dot(w2.T,dz2) * a1*(1-a1)
    dw1 = np.dot(dz1,x.T)/tot
    dw1 = np.reshape(dw1,w1.shape)

    dw2 = np.reshape(dw2,w2.shape)
    return dz2,dw2,dz1,dw1
```




```
#Defining Prediction method
def predict(w1,w2,input):
    z1,a1,z2,a2 = frwd_prop(w1,w2,mlp_test)
    a2 = np.squeeze(a2)
    if a2>=0.5:
        print("For input", [i[0] for i in input], "output is 1")
    else:
        print("For input", [i[0] for i in input], "output is 0")
```

```
#Calling Network Model
mlp_test = np.array([[1],[0]])
predict(w1,w2,mlp_test)
mlp_test = np.array([[0],[0]])
predict(w1,w2,mlp_test)
mlp_test = np.array([[0],[1]])
predict(w1,w2,mlp_test)
mlp_test = np.array([[1],[1]])
predict(w1,w2,mlp_test)
```

```
For input [1, 0] output is 1
For input [0, 0] output is 0
For input [0, 1] output is 1
For input [1, 1] output is 0
```

Lab Exercises:

1. To reinforce the perceptron, apply learning procedure for XOR Gate. The gate returns 0 if and only if both inputs are 0. Do not hesitate to change the initial weights and learning rate values.

3. Can this form generalized the same algorithm for nonlinear problem XNOR gate , why or why not?

_____
_____
_____
_____
_____

Conclusion:

_____
_____
_____
_____
_____
_____

K-Nearest Neighbors (KNN)

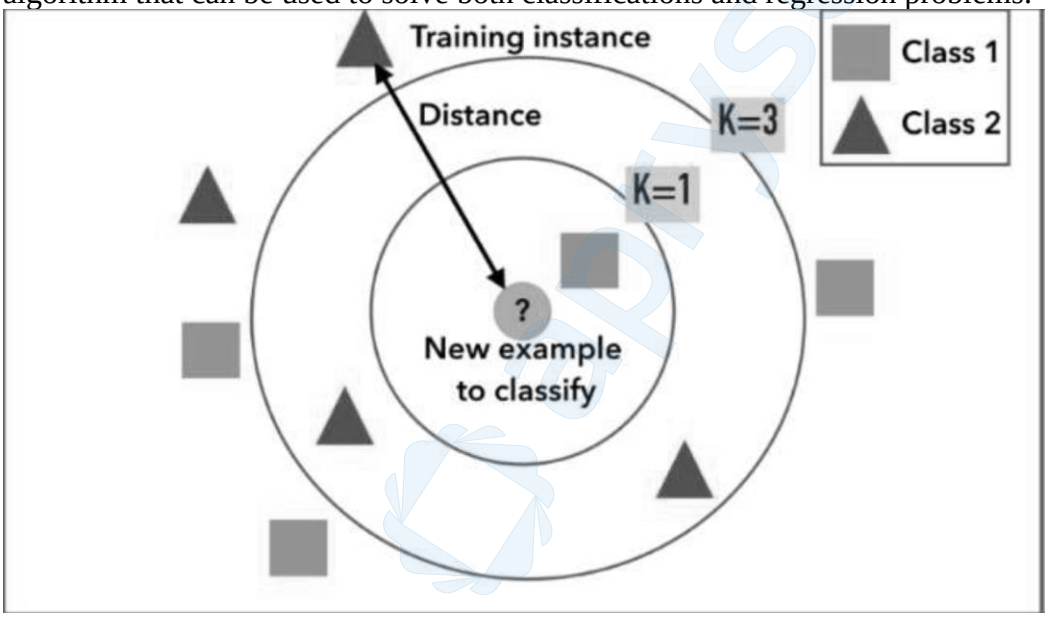
Objective:

To perform the K-Nearest Neighbor algorithm and determine the outcome of the algorithm performance.

Background and Procedure:

K-Nearest Neighbor

The k-nearest neighbors algorithm is a simple easy to implement supervised machine learning algorithm that can be used to solve both classifications and regression problems.



Implementation of KNN Algorithm using IRIS dataset

- **About IRIS Data Set**

This dataset contains 3 classes of 50 instances each, which refers to a type of iris plant.

Attribute Information:

1. Sepal length in cm
2. Sepal width in cm
3. Petal length in cm
4. Petal width in cm
5. Species: “Iris Setosa”, “Iris Versicolour”, “Iris Virginica”

```
import pandas as pd
from sklearn.datasets import load_iris
iris_dataset = load_iris() #loading iris dataset uaing sklearn
```

```
iris_dataset.feature_names
```

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

```
iris_dataset.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

```
df = pd.DataFrame(iris_dataset.data,columns=iris_dataset.feature_names)
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

```
df['target'] = iris_dataset.target
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
df['flower_name'] =df.target.apply(lambda x: iris_dataset.target_names[x])
df.head(10)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa
5	5.4	3.9	1.7	0.4	0	setosa
6	4.6	3.4	1.4	0.3	0	setosa
7	5.0	3.4	1.5	0.2	0	setosa
8	4.4	2.9	1.4	0.2	0	setosa

```
df[df.target==1].head() #display data contanining target value 1
#df[df.target==2].head() #display data contanining target value 2
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
50	7.0	3.2	4.7	1.4	1	versicolor
51	6.4	3.2	4.5	1.5	1	versicolor
52	6.9	3.1	4.9	1.5	1	versicolor
53	5.5	2.3	4.0	1.3	1	versicolor
54	6.5	2.8	4.6	1.5	1	versicolor

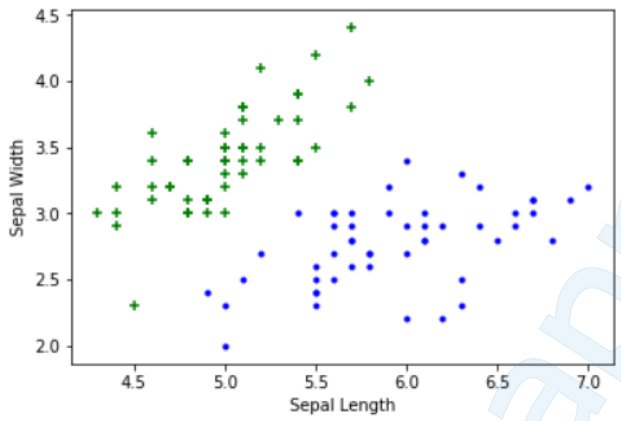
```
df[45:55] #checking range of target values
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
45	4.8	3.0	1.4	0.3	0	setosa
46	5.1	3.8	1.6	0.2	0	setosa
47	4.6	3.2	1.4	0.2	0	setosa
48	5.3	3.7	1.5	0.2	0	setosa
49	5.0	3.3	1.4	0.2	0	setosa
50	7.0	3.2	4.7	1.4	1	versicolor
51	6.4	3.2	4.5	1.5	1	versicolor
52	6.9	3.1	4.9	1.5	1	versicolor
53	5.5	2.3	4.0	1.3	1	versicolor
54	6.5	2.8	4.6	1.5	1	versicolor

```
df0 = df[:50]
df1 = df[50:100]
df2 = df[100:]
```

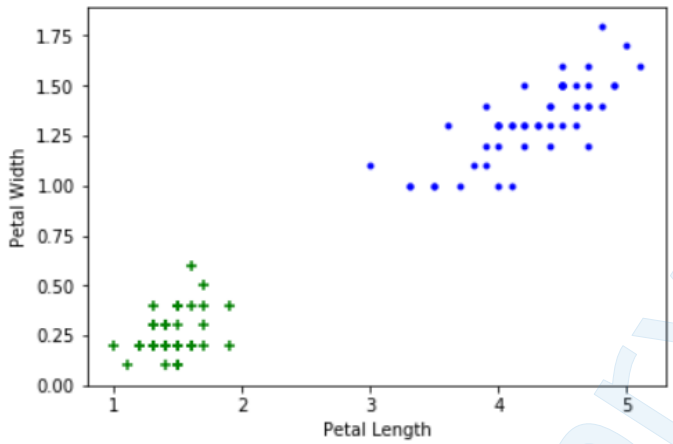
```
import matplotlib.pyplot as plt
%matplotlib inline

#Sepal length vs Sepal Width (Setosa vs Versicolor)
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.scatter(df0['sepal length (cm)'], df0['sepal width (cm)'],color="green",marker='+')
plt.scatter(df1['sepal length (cm)'], df1['sepal width (cm)'],color="blue",marker='.')
```

```
<matplotlib.collections.PathCollection at 0x24ce7801908>
```



```
#Petal length vs Pepal Width (Setosa vs Versicolor)
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.scatter(df0['petal length (cm)'], df0['petal width (cm)'],color="green",marker='+')
plt.scatter(df1['petal length (cm)'], df1['petal width (cm)'],color="blue",marker='.')
```

```
<matplotlib.collections.PathCollection at 0x24ce886fa20>
```



```
#Creating traning and testing dataframe
#Train test split

from sklearn.model_selection import train_test_split
X = df.drop(['target','flower_name'], axis='columns')
y = df.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

```
len(X_train)
```

```
105
```

```
len(X_test)
```

```
45
```

```
#creating KNN (K Neighrest Neighbour Classifier)
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=10)
```

```
knn.fit(X_train, y_train) #learning and prediction
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=10, p=2,
           weights='uniform')
```

```
knn.score(X_test, y_test) #check prediction/performance score
```

```
0.9777777777777777
```

```
#knn.predict([[4.8,3.0,1.5,0.3]]) #passing test values to predict the species
predicted_value = knn.predict([[6.5,2.8,4.6,1.5]]) #passing test values to predict the species

print('KNN Species Prediction:', predicted_value)
```

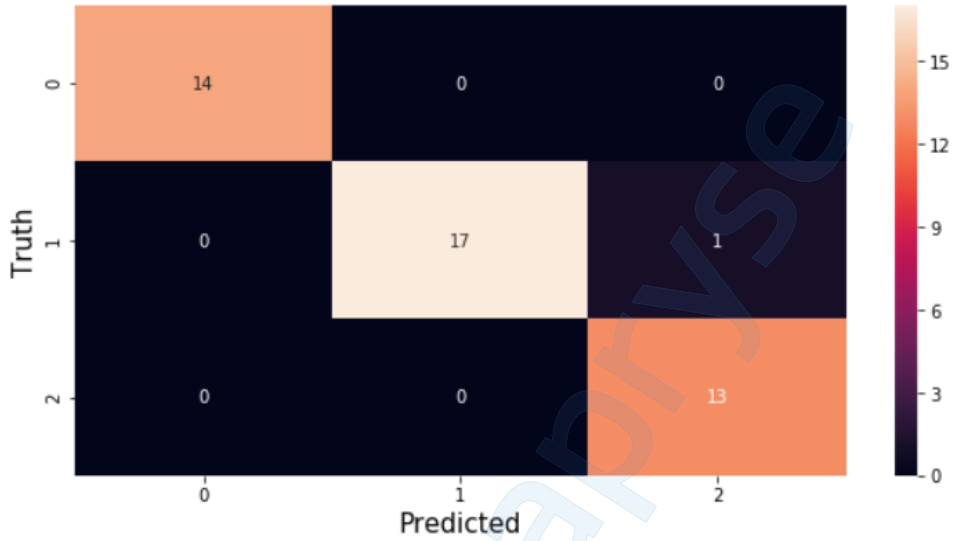
```
KNN Species Prediction: [1]
```

```
#Plotting Confusion Matrix

from sklearn.metrics import confusion_matrix
y_pred = knn.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[14,  0,  0],
       [ 0, 17,  1],
       [ 0,  0, 13]], dtype=int64)
```

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sn
plt.figure(figsize=(10,5))
sn.heatmap(cm, annot=True)
plt.xlabel('Predicted', fontsize = 15)
plt.ylabel('Truth', fontsize = 15)
```

```
Text(69,0.5,'Truth')
```




```
#Generating classification report for precesion, recall and f1-score for each classes

from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
             precision    recall  f1-score   support

          0       1.00      1.00      1.00        14
          1       1.00      0.94      0.97        18
          2       0.93      1.00      0.96        13

avg / total       0.98      0.98      0.98        45
```

Lab Exercises:

1. Perform the given KNN algorithm and analyze result prediction.

4. Can this learning generalized the same algorithm for unsupervised problem, why or why not?

_____
_____
_____
_____
_____

Conclusion:

_____
_____
_____
_____
_____
_____

Correlation and Regression:

Objective:

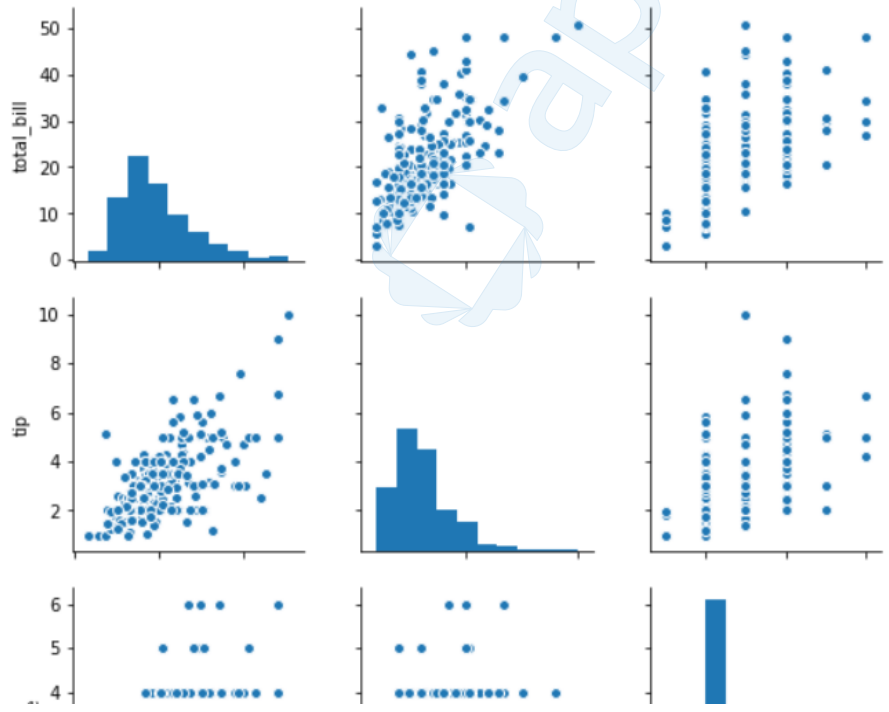
To perform the correlation and regression algorithms and determine the machine learning mechanism of the algorithm performance.

Background and Procedure:

Correlation

Statistics and data science are often concerned about the relationships between two or more variables (or features) of a dataset. Each data point in the dataset is an observation, and the features are the properties or attributes of those observations.

```
#correlation example using seaborn
import matplotlib.pyplot as plt
import seaborn as sb

#example of the tips data set available in seaborn python library
dataframe = sb.load_dataset('tips')
sb.pairplot(dataframe, kind="scatter")
plt.show()
```



```python
#Calculating correlation values - pandas

import pandas as pd

x = pd.Series(range(1,20))
y = pd.Series([5,45,88,3,5,44,9,62,11,50])

print(x.corr(y))
#print(y.corr(x))
```

```
0.058065327411248945
```

```python
#Calculating correlation values - numpy

import numpy as np
#x = np.arange(1,11)
x = np.arange(0,10)
y = np.array([5,45,88,3,5,44,9,62,11,50])
find_correlation = np.corrcoef(x,y)

print('Values of x', x)
print('Values of y', y)
print('\n Correlation Coefficient: \n', find_correlation)
# #print('\n Correlation Coefficient: \n', find_correlation[0,1])
```

```
Values of x [0 1 2 3 4 5 6 7 8 9]
Values of y [ 5 45 88  3  5 44  9 62 11 50]

 Correlation Coefficient:
 [[1.         0.05806533]
 [0.05806533 1.        ]]
```

Regression

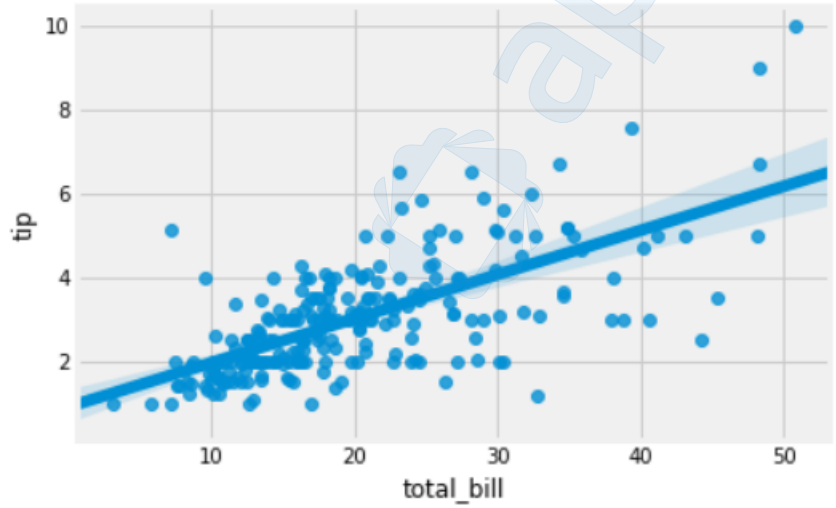
Regression searches for relationships among variables. Linear regression is used to identify the relationship between a dependent variable and one or more independent variables and is typically leveraged to make predictions about future outcomes.

When there is only one independent variable and one dependent variable, it is known as simple linear regression. As the number of independent variables increases, it is referred to as multiple linear regression. For each type of linear regression, it seeks to plot a line of best fit, which is calculated through the method of least squares.

However, unlike other regression models, this line is straight when plotted on a graph. Linear regression is leveraged when dependent variables are continuous, whereas logistical regression is selected when the dependent variable is categorical, meaning they have binary outputs, such as "true" and "false" or "yes" and "no."

While both regression models seek to understand relationships between data inputs, logistic regression is mainly used to solve binary classification problems, such as spam identification.

```
#regression example using seaborn
import seaborn as sb
from matplotlib import pyplot as plt
dataframe = sb.load_dataset('tips')
sb.regplot(x = "total_bill", y = "tip", data = dataframe)
plt.show()
```



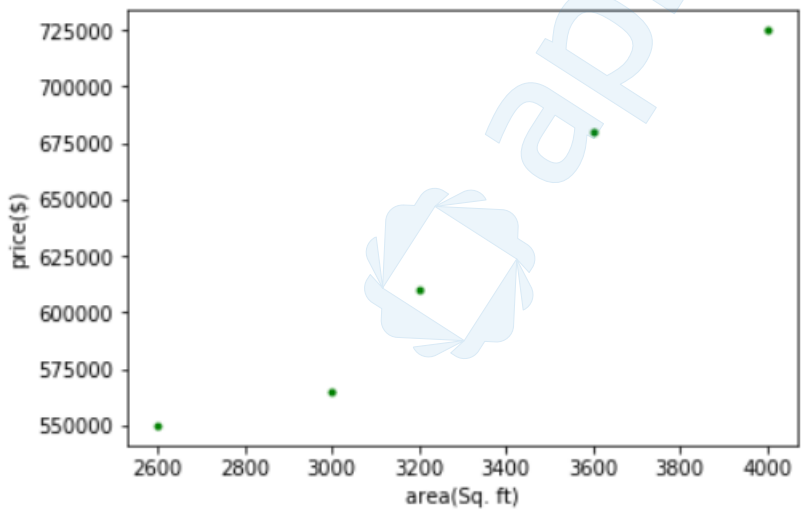
```
import pandas as pd
import numpy as np
from sklearn import linear_model #scikit learn python library, using linear_model for linear regression python model
import matplotlib.pyplot as plt
```

```
#loading data
dataframe = pd.read_csv("homeprices.csv")
dataframe
```

	area	price
0	2600	550000
1	3000	565000
2	3200	610000
3	3600	680000
4	4000	725000

```
#for displaying distribution of datapoints, using scatter plot
#%matplotlib inline
plt.xlabel('area(Sq. ft)')
plt.ylabel('price($)')
plt.scatter(dataframe.area,dataframe.price,color='green',marker='.')
```

```
<matplotlib.collections.PathCollection at 0x2057707fa20>
```



```
#creating object for linear regression model
linear_Reg_Obj = linear_model.LinearRegression()

#training linear regression model using available datapoints
linear_Reg_Obj.fit(dataframe[['area']], dataframe.price) #passign arguments to 2D array
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

```
#predicting price
#by calculating cooefficient and intercept
linear_Reg_Obj.predict(4500)
```

```
array([791660.95890411])
```

```
#displaying value of coefficient
linear_Reg_Obj.coef_
```

```
array([135.78767123])
```

```
#displaying value of y intercept
linear_Reg_Obj.intercept_
```

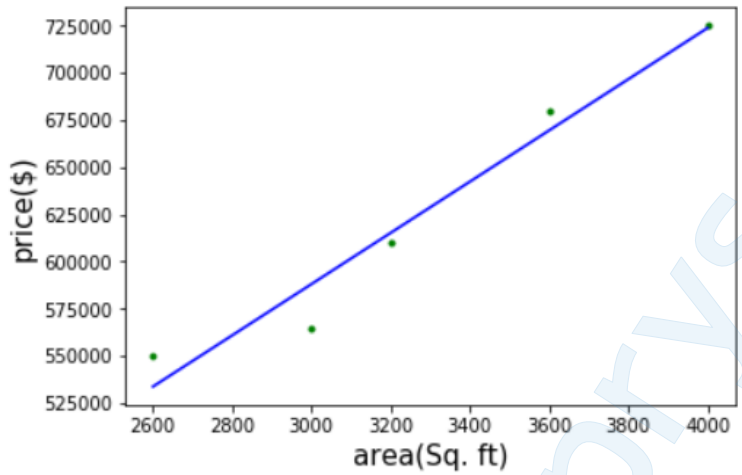
```
180616.43835616432
```

```
#understanding through linear equation y=mx+b
#here the equation become y=m*area+b i-e y=coef*area+intercept

135.78767123*4500+180616.43835616432
#which is same as linear_Reg_Obj.predict(4500)
```

```
791660.9588911643
```

```
#Plotting regression line
plt.xlabel('area(Sq. ft)', fontsize= 15)
plt.ylabel('price($)', fontsize= 15)
plt.scatter(dataframe.area,dataframe.price,color='green',marker='.')
plt.plot(dataframe.area, linear_Reg_Obj.predict(dataframe[['area']]),color='blue')
```

```
[<matplotlib.lines.Line2D at 0x20577373dd8>]
```



Creating Dataframe file for prediction.csv

```
#PREDICTING PRICES USING DATAFRAME

#creating dataframe to generate CSV file with list of home price predictions
new_dataframe = pd.read_csv("areas.csv")
new_dataframe.head(5)
```

	area
0	1000
1	1500
2	2300
3	3540
4	4120

```
#using regression model to predict prices
linear_Reg_Obj.predict(new_dataframe)
```

```
array([ 316404.10958904,  384297.94520548,  492928.08219178,
        661304.79452055,  740061.64383562,  799808.21917808,
        926090.75342466,  650441.78082192,  825607.87671233,
        492928.08219178, 1402705.47945205, 1348390.4109589 ,
       1144708.90410959])
```

```
#assigning the new predicted prices into a variable 'predicted_homeprices'

predicted_homeprices = linear_Reg_Obj.predict(new_dataframe)

#creating a new column in the existing dataframe
new_dataframe['prices'] = predicted_homeprices
```

```
#calling the updated dataframe
new_dataframe
```

	area	prices
0	1000	3.164041e+05
1	1500	3.842979e+05
2	2300	4.929281e+05
3	3540	6.613048e+05
4	4120	7.400616e+05
5	4560	7.998082e+05
6	5490	9.260908e+05
7	3460	6.504418e+05
8	4750	8.256079e+05
9	2300	4.929281e+05
10	9000	1.402705e+06
11	8600	1.348390e+06
12	7100	1.144709e+06

```
#creating a new file 'prediction' and exporting values to prediction.csv file
#new_dataframe.to_csv('prediction.csv')

#to remove index column from the dataframe or csv file
new_dataframe.to_csv('prediction.csv', index= False)
```

Lab Exercises:

1. Perform correlation problem define in the manual.

2. Implement linear regression technique for a sample dataset for house price prediction.

Conclusion:

K-Means Clustering

Objective:

To perform the K-Means clustering algorithm and determine the outcome of the algorithm performance.

Background and Procedure:

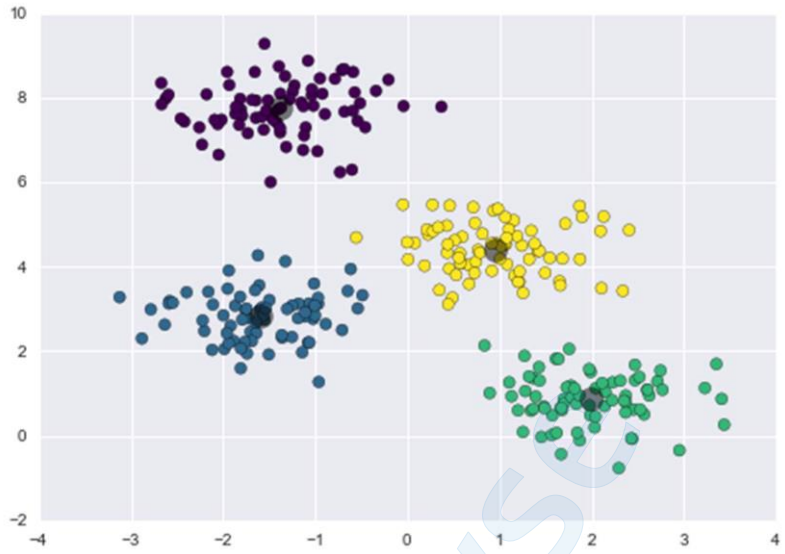
Clustering

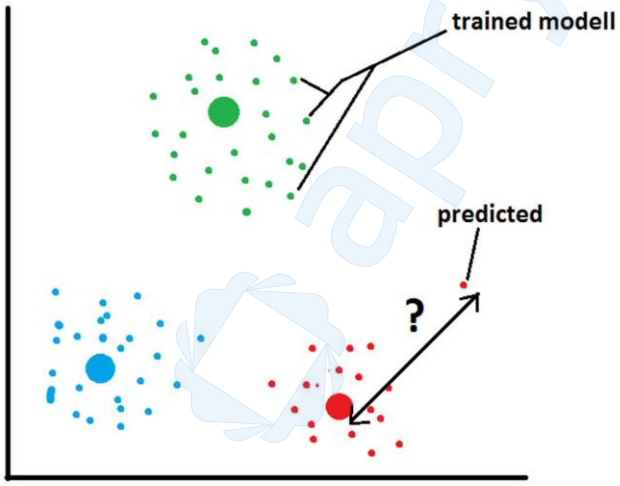
Clustering is a set of techniques used to partition data into groups, or clusters. Clusters are loosely defined as groups of data objects that are more similar to other objects in their cluster than they are to data objects in other clusters. In practice, clustering helps identify two qualities of data:

- Meaningfulness
- Usefulness

Meaningful clusters expand domain knowledge. For example, in the medical field, researchers applied clustering to gene expression experiments. The clustering results identified groups of patients who respond differently to medical treatments.
Useful clusters, on the other hand, serve as an intermediate step in a data pipeline. For example, businesses use clustering for customer segmentation. The clustering results segment customers into groups with similar purchase histories, which businesses can then use to create targeted advertising campaigns.

When we are working with huge volumes of data, it makes sense to partition the data into logical groups and doing the analysis. We can use Clustering to make the data into groups with the help of several algorithms like K-Means.





K MEANS CLUSTERING ALGORITHM STEPS

- Choose a random number of centroids in the data. i.e k=3.
- Choose the same number of random points on the 2D canvas as centroids.
- Calculate the distance of each data point from the centroids.
- Allocate the data point to a cluster where its distance from the centroid is minimum.
- Recalculate the new centroids.
- Recalculate the distance from each data point to new centroids.
- Repeat the steps from point 3, till no data point change its cluster.

```
from sklearn.cluster import KMeans
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot as plt
%matplotlib inline

df = pd.read_csv("income.csv")
df.head()

plt.scatter(df.Age,df['Income($)'])
plt.xlabel('Age')
plt.ylabel('Income($)')

km = KMeans(n_clusters=3)
y_predicted = km.fit_predict(df[['Age','Income($)']])
y_predicted

df['cluster']=y_predicted
df.head()

km.cluster_centers_

df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]
plt.scatter(df1.Age,df1['Income($)'],color='green')
plt.scatter(df2.Age,df2['Income($)'],color='red')
plt.scatter(df3.Age,df3['Income($)'],color='black')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marker='*',label='centroid')
plt.xlabel('Age')
plt.ylabel('Income ($)')
plt.legend()
```

```
scaler = MinMaxScaler()
scaler.fit(df[['Income($)']])
df['Income($)'] = scaler.transform(df[['Income($)']])
scaler.fit(df[['Age']])
df['Age'] = scaler.transform(df[['Age']])

df.head()

plt.scatter(df.Age,df['Income($)'])

km = KMeans(n_clusters=3)
y_predicted = km.fit_predict(df[['Age','Income($)']])
y_predicted

df['cluster']=y_predicted
df.head()

km.cluster_centers_

df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]
plt.scatter(df1.Age,df1['Income($)'],color='green')
plt.scatter(df2.Age,df2['Income($)'],color='red')
plt.scatter(df3.Age,df3['Income($)'],color='black')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marker='*',label='centroid')
plt.legend()
```

```
#Elbow Plot
sse = []
k_rng = range(1,10)
for k in k_rng:
    km = KMeans(n_clusters=k)
    km.fit(df[['Age','Income($)']])
    sse.append(km.inertia_)

plt.xlabel('K')
plt.ylabel('Sum of squared error')
plt.plot(k_rng,sse)
```

Lab Exercises:

1. Implement cluster algorithm Kmeans for income dataset of salaried employees.

2. What main difference does Kmeans has when compared to KNN?

_____
_____
_____
_____
_____

Conclusion:

_____
_____
_____
_____
_____
_____

Naïve Bayes

Objective:

To perform naïve bayes algorithm and determine the outcome of the algorithm performance.

Background and Procedure:

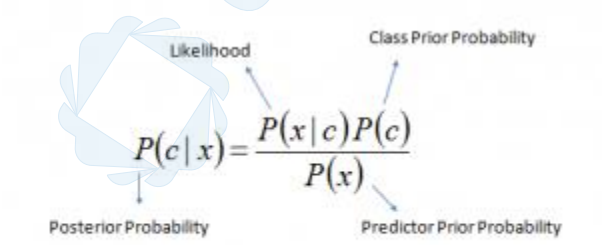
Naïve Bayes

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'.

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:



$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Above,

P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes).
P(c) is the prior probability of class.
P(x|c) is the likelihood which is the probability of predictor given class.
P(x) is the prior probability of predictor.

```
import pandas as pd
```

```
df = pd.read_csv("spam.csv")
df.head()
```

	Category	Message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

```
df.groupby('Category').describe()
```

	Message				
	count	unique	top		freq
Category					
ham	4825	4516	Sorry, I'll call later		30
spam	747	641	Please call our customer service representativ...		4

```
df['spam']=df['Category'].apply(lambda x: 1 if x=='spam' else 0)
df.head()
```

	Category	Message	spam
0	ham	Go until jurong point, crazy.. Available only ...	0
1	ham	Ok lar... Joking wif u oni...	0
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	1
3	ham	U dun say so early hor... U c already then say...	0
4	ham	Nah I don't think he goes to usf, he lives aro...	0

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df.Message,df.spam)
```

```
from sklearn.feature_extraction.text import CountVectorizer
v = CountVectorizer()
X_train_count = v.fit_transform(X_train.values)
X_train_count.toarray()[:2]
```

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

```
from sklearn.naive_bayes import MultinomialNB
model = MultinomialNB()
model.fit(X_train_count,y_train)
```

```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

```
emails = [
    'Hey mohan, can we get together to watch footbal game tomorrow?',
    'Upto 20% discount on parking, exclusive offer just for you. Dont miss this reward!'
]
emails_count = v.transform(emails)
model.predict(emails_count)
```

```
array([0, 1], dtype=int64)
```

```
X_test_count = v.transform(X_test)
model.score(X_test_count, y_test)
```

```
0.9827709978463748
```

Sklearn Pipeline

```
from sklearn.pipeline import Pipeline
clf = Pipeline([
    ('vectorizer', CountVectorizer()),
    ('nb', MultinomialNB())
])
```

```
clf.fit(X_train, y_train)
```

```
Pipeline(memory=None,
     steps=[('vectorizer', CountVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
        strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
        tokenizer=None, vocabulary=None)), ('nb', MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True))])
```

```
from sklearn.pipeline import Pipeline
clf = Pipeline([
    ('vectorizer', CountVectorizer()),
    ('nb', MultinomialNB())
])
```

```
clf.fit(X_train, y_train)
```

```
Pipeline(memory=None,
     steps=[('vectorizer', CountVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
        strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
        tokenizer=None, vocabulary=None)), ('nb', MultinomialNB(alpha=1.0, class_prior=None, fit_prior=
```

```
clf.score(X_test,y_test)
```

```
0.9827709978463748
```

```
clf.predict(emails)
```

```
array([0, 1], dtype=int64)
```

Lab Exercises:

1. Implement naïve bayes algorithm technique to detect spam email.

Conclusion:

Genetic Algorithm

Objective:

To perform genetic algorithm and determine the outcome of the algorithm performance.

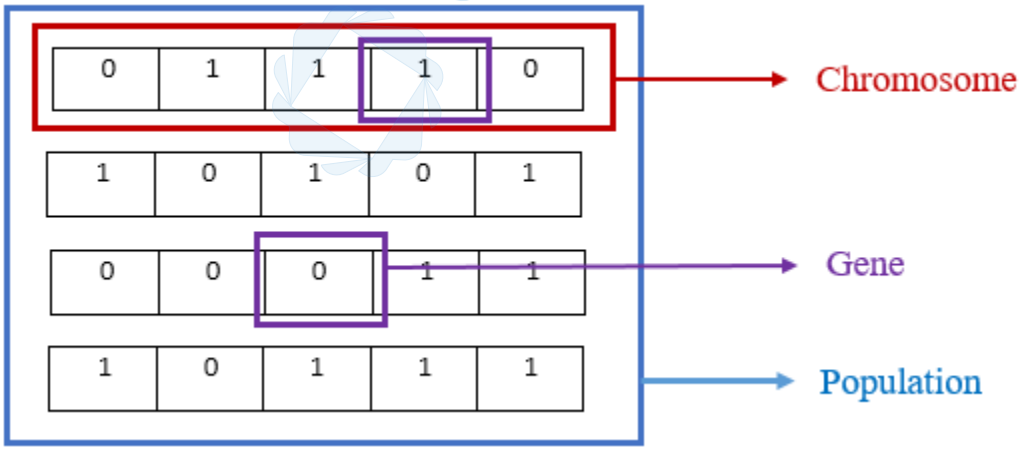
Background and Procedure:

Genetic Algorithm

The genetic algorithm is a search-based optimization technique. It is frequently used to find the optimal or nearest optimal solution
In genetic we will use some biological terms such as population, chromosome, gene, selection, crossover, mutation. Now, first of all, let's try to understand what these terms mean.

Population, Chromosome, Gene

At the beginning of this process, we need to initialize some possible solutions to this problem. The population is a subset of all possible solutions to the given problem. In another way, we can say that the population is a set of chromosomes. A chromosome is one of that solution to that current problem. And each chromosome is a set of genes.
For simplicity, We can describe a chromosome as a string. So, we can say that a population is a collection of some string(each character is a binary value, either 0 or 1). And each character of the string is a gene.



For starting the process of the genetic algorithm, we first need to initialize the population. We can initialize the population in two ways. The first one is random and the second one is heuristical. It is always better to start the algorithm with some random population.

Fitness Function

After initializing the population, we need to calculate the fitness value of these chromosomes. Now the question is what this fitness function is and how it calculates the fitness value.
As an example, let consider that we have an equation, $f(x) = -x^2 + 5$ .We need the solution for which it has the maximum value and the constraint is $0 \leq x \leq 31$.

Now, let consider that we have a random population of four chromosomes like below. The length of our chromosome is 5 as $2^5=32$ and $0 \leq x \leq 31$.

0	1	1	1	0
1	0	1	0	1
0	0	0	1	1
1	0	1	1	1

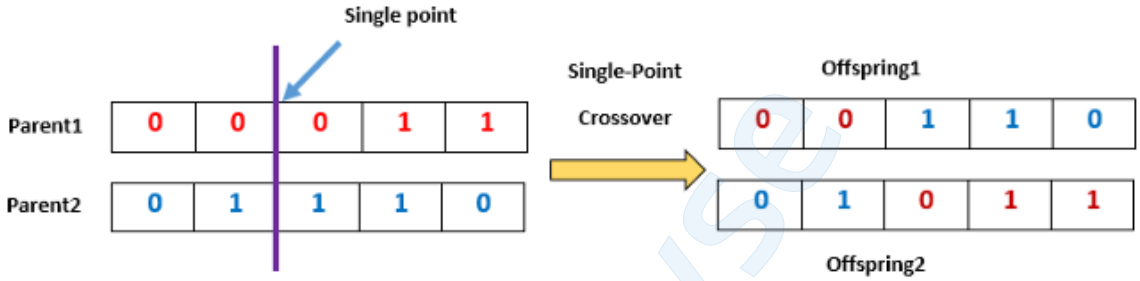
Our fitness function will calculate the functional value of each chromosome as stated in the problem statement :
For the 1st chromosome, 01110 means 14 in integer. So, f(x) = -(14*14) + 5 = -191.
For the 2nd chromosome, 10101 means 21 in integer. So, f(x) = -(21*21) + 5 = -436.
For the 3rd chromosome, 00011 means 3 in integer. So, f(x) = -(3*3) + 5 = -4.
For the 4th chromosome, 10111 means 23 in integer. So, f(x) = -(23*23) + 5 = -524.

Parent Selection

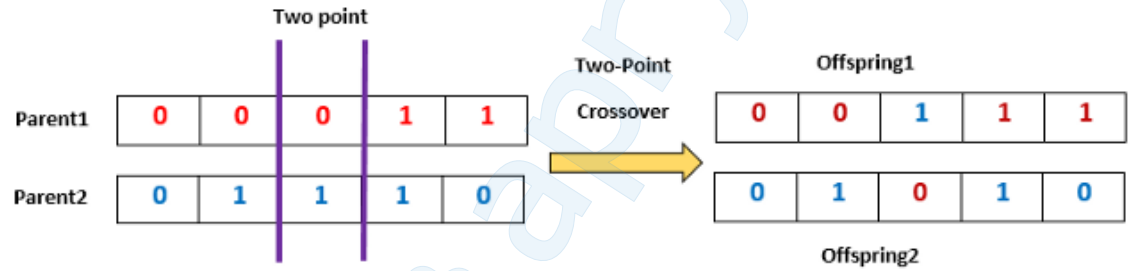
Parent selection is done by using the fitness values of the chromosomes calculated by the fitness function. Based on these fitness values we need to select a pair chromosomes with the highest fitness value.
There are many ways for fitness calculation like Roulette wheel selection, rank selection.
In Roulette wheel selection, the chromosome with the highest fitness value has the maximum possibility to be selected as a parent. But in this selection process, a lower can be selected.
In rank selection, chromosomes are ranked based on their fitness values from higher to lower. As an example, According to those fitness values calculated above, we can rank those chromosomes from higher to lower like 3rd>1st>2nd>4th. So, in the selection phase, 3rd and 1st chromosomes will be selected based on the fitness valued calculated from the fitness function.

Crossover

Crossover is used to vary the programming of the chromosomes from one generation to another by creating children or offsprings. Parent chromosomes are used to create these offsprings(generated chromosomes).
To create offsprings, there are some ways like a single-point crossover, two or multi-point crossover.
For a single point crossover, first, we need to select a point and then exchange these portions divided by this point between parent chromosomes to create offsprings. You can use the color combination for easy understanding.



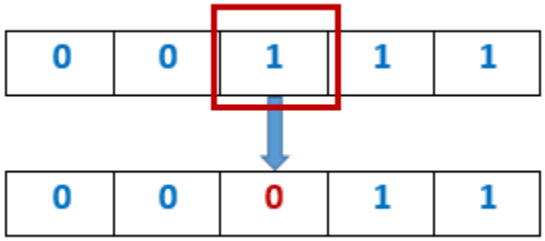
For a two-point crossover, we need to select two points and then exchange the bits.



Finally, these new offsprings are added to the population.

Mutation

Mutation brings diversity to the population. There are different kinds of mutations like Bit Flip mutation, Swap mutation, Inversion mutation, etc. These are so so simple.
In Bit Flip mutation, Just select one or more bits and then flip them. If the selected bit is 0 then turn it to 1 and if the selected bit is 1 then turn it to 0.



In Swap Bit mutation, select two bits and just swap them.

In inverse mutation, just inverse the bits.

Implementation Of Genetic Algorithm In Python

Let's try to implement the genetic algorithm in python for function optimization.

Problem Statement

Let consider that we have an equation, $f(x) = -x^2 + 5$. We need the solution for which it has the maximum value and the constraint is $0 \leq x \leq 31$. To select an initial population use the probability 0.2.

Initial Population

Random Initialization is better than heuristic initialization. Therefore, here random initialization is used for population initialization.

```
#initialize population
import random
best=-100000
populations =([[random.randint(0,1) for x in range(6)] for i in range(4)])
print(type(populations))
parents=[]
new_populations = []
print(populations)
```

```
<class 'list'>
[[1, 1, 0, 0, 1, 0], [1, 0, 1, 0, 1, 0], [0, 1, 0, 0, 1, 1], [0, 1, 0, 1, 1, 0]]
```

Fitness Function

The fitness function calculates the fitness value of chromosomes. The functionality of the fitness function depends on the problem's requirements.

```
#fitness score calculation .....
def fitness_score() :
    global populations,best
    fit_value = []
    fit_score=[]
    for i in range(4) :
        chromosome_value=0

        for j in range(5,0,-1) :
            chromosome_value += populations[i][j]*(2**(5-j))
        chromosome_value = -1*chromosome_value if populations[i][0]==1 else chromosome_value
        print(chromosome_value)
        fit_value.append(-(chromosome_value**2) + 5 )
    print(fit_value)
    fit_value, populations = zip(*sorted(zip(fit_value, populations) , reverse = True))
    best= fit_value[0]

fitness_score()
```

```
-18
-10
19
22
[-319, -95, -356, -479]
```

Selection

Fittest chromosomes are selected based on the fitness scores. Here rank selection process is used.

```
def selectparent():
    global parents
    #global populations , parents
    parents=populations[0:2]
    print(type(parents))
    print(parents)
selectparent()
```

```
<class 'tuple'>
([1, 0, 1, 0, 1, 0], [1, 1, 0, 0, 1, 0])
```

Crossover

After selecting the fittest parents, a crossover is required to generate offsprings. Here, the single-point crossover is used.

```
def crossover() :
    global parents

    cross_point = random.randint(0,5)
    parents=parents + tuple([(parents[0][0:cross_point +1] +parents[1][cross_point+1:6])])
    parents =parents+ tuple([(parents[1][0:cross_point +1] +parents[0][cross_point+1:6])])

    print(parents)
```

Mutation

After crossover is done, a mutation is done for maintaining the diversity from one generation to another. Here, we will single point bit-flip mutation.

```
def mutation() :
    global populations, parents
    mute = random.randint(0,49)
    if mute == 20 :
        x=random.randint(0,3)
        y = random.randint(0,5)
        parents[x][y] = 1-parents[x][y]
    populations = parents
    print(populations)
```

We need to iterate the whole process multiple times until we find our best solution.

```
for i in range(1000) :
    fitness_score()
    selectparent()
    crossover()
    mutation()
print("best score :")
print(best)
print("sequence........")
print(populations[0])
```

```
-10
-18
19
22
[-95, -319, -356, -479]
<class 'tuple'>
([1, 0, 1, 0, 1, 0], [1, 1, 0, 0, 1, 0])
([1, 0, 1, 0, 1, 0], [1, 1, 0, 0, 1, 0], [1, 0, 1, 0, 1, 0], [1, 1, 0, 0, 1, 0])
([1, 0, 1, 0, 1, 0], [1, 1, 0, 0, 1, 0], [1, 0, 1, 0, 1, 0], [1, 1, 0, 0, 1, 0])
-10
-18
-10
-18
[-95, -319, -95, -319]
<class 'tuple'>
([1, 0, 1, 0, 1, 0], [1, 0, 1, 0, 1, 0])
([1, 0, 1, 0, 1, 0], [1, 0, 1, 0, 1, 0], [1, 0, 1, 0, 1, 0], [1, 0, 1, 0, 1, 0])
([1, 0, 1, 0, 1, 0], [1, 0, 1, 0, 1, 0], [1, 0, 1, 0, 1, 0], [1, 0, 1, 0, 1, 0])
-10
  ~
  [5, 5, 5, 5]
  <class 'tuple'>
  ([1, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0])
  ([1, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0])
  ([1, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0])
  best score :
  5
  sequence........
  [1, 0, 0, 0, 0, 0]
```

Lab Exercises:

2. Implement genetic algorithm performing all steps as applied in the coding example.

3. What is mutation and fitness in genetic algorithm?

Conclusion:

Search Algorithm

Objective:

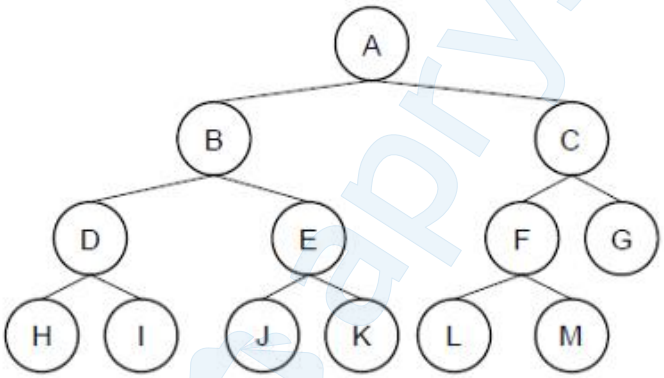
To perform depth first search, breath first search and classical AI algorithm techniques.

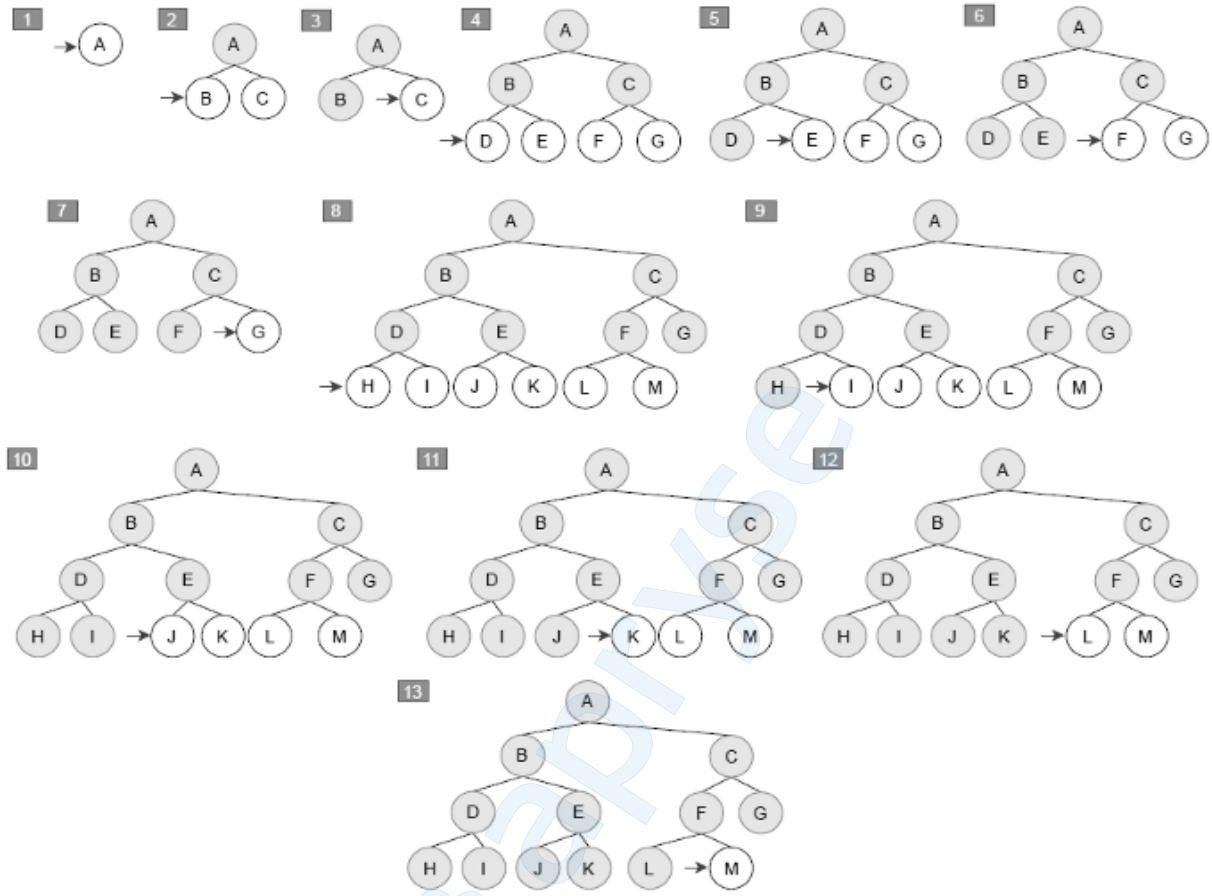
Background and Procedure:

Breadh First Search

One of the foundation search algorithms called Blind Search.

The search algorithm is an algorithm to retrieve information stored within some data structure, or calculated in the search space of a problem domain. Breadth-first search is an algorithm for traversing or searching tree or graph data structures. It starts at the root node and explores all nodes at the present depth before moving on to the nodes at the next depth level. In other words, it expands the shallowest unexpanded node which can be implemented by a First-In-First-Out (FIFO) queue. Let's go through an example with the following GRAPH:





From the above step by step expansion diagram, we can see that the BFS algorithm prioritizes edges closest to the starting vertex to perform a search.

Now, let's evaluate this algorithm:
Denote d as the depth of the least-cost solution and b as the maximum branching factor of the search tree or graph. Assume a hypothetical state space where every node can be expanded into b new nodes, solution of path-length d:

Time Complexity: How long does it take to find a solution?
$1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$
Space Complexity: Maximum number of nodes in memory
Keeps every node in memory = $O(b^d)$
Completeness: Does it always find a solution if one exists?
Yes
Optimality: Does it always find the best (least-cost) solution?
Yes, when all steps cost equally.

The diagram is a schematic representation of the graph with vertices

V={A, B, C, D, E, F, G, H, I, J, K, L, M}, E={{A, B}, {A, C}, {B, D}, {B, E}, {C, F}, {C, G}, {D, H}, {D, I}, {E, J}, {E, K}, {F, L}, {F, M}}

BFS Implementation

Creating the function that takes in edges of the GRAPH which outputs the adjacency list for undirected graph

```
from collections import defaultdict

# Generate adjacency list for undirected graph
def generateAdjacencyLst(edges):
    adjacencyList = defaultdict(list)
    for u, v in edges:
        adjacencyList[u].append(v)
        adjacencyList[v].append(u)
    return adjacencyList

edges = [['A', 'B'], ['A', 'C'], ['B', 'D'], ['B', 'E'],
         ['C', 'F'], ['C', 'G'], ['D', 'H'], ['D', 'I'],
         ['E', 'J'], ['E', 'K'], ['F', 'L'], ['F', 'M']]
adjacencyList = generateAdjacencyLst(edges)
adjacencyList
```

```
defaultdict(list,
            {'A': ['B', 'C'],
             'B': ['A', 'D', 'E'],
             'C': ['A', 'F', 'G'],
             'D': ['B', 'H', 'I'],
             'E': ['B', 'J', 'K'],
             'F': ['C', 'L', 'M'],
             'G': ['C'],
             'H': ['D'],
             'I': ['D'],
             'J': ['E'],
             'K': ['E'],
             'L': ['F'],
             'M': ['F']})
```

Creating the function that takes in the adjacency list and starting vertex which output the sequence of BFS search

```
# find bfs traversal from starting vertex
def bfs(adjacencyList, vertex):
    visitedSet = set()
    queue = []
    visitedSet.add(vertex)
    queue.append(vertex)

    result = []
    while queue:
        v = queue[0]
        result.append(v)
        queue = queue[1:]
        for neighbor in adjacencyList[v]:
            if neighbor not in visitedSet:
                visitedSet.add(neighbor)
                queue.append(neighbor)
    return result

bfs(adjacencyList, 'A')
```

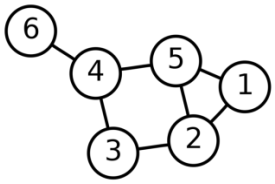
```
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M']
```

Depth First Search

Depth-first search is an algorithm for traversing or searching tree or graph data structures [2]. Before explaining the DFS algorithm, let's introduce the graph data structure. A graph G is a pair (V, E), where V is a finite set and E is a set of binary relations on V.

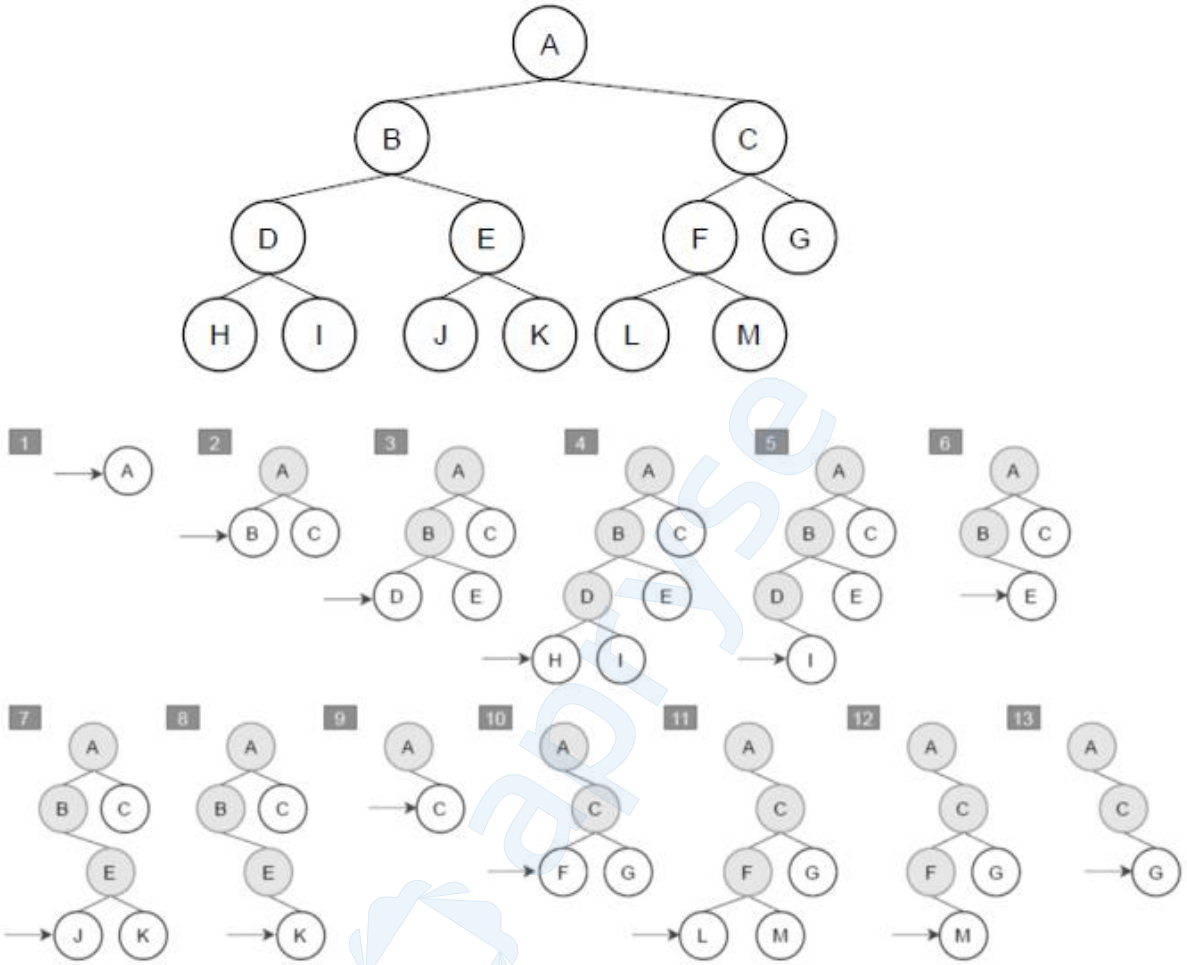
V is called the vertex set and its elements are vertices.

E is called the edge set and its elements are called edges. An edge is represented by a pair of vertices.



The diagram is a schematic representation of the graph with vertices V={1, 2, 3, 4, 5, 6}, E={{1, 2},{1, 5},{2, 3},{2, 5},{3, 4},{4, 5},{4, 6}}

It starts at the root node and expands the deepest unexpanded node, backtracks only when no more expansion. Let's go through an example with the following GRAPH:



From the above step by step expansion diagram, we can see that the DFS algorithm prioritizes edges along the path to perform a search.

Now, let's evaluate this algorithm:
Denote m as the maximum depth of the state space and b as the maximum branching factor of the search tree or graph

Completeness:

- Infinite-depth spaces: No
- Finite-depth spaces with loops: No
- Finite-depth spaces with repeated-state checking: Yes
- Finite-depth spaces without loops: Yes
- Time Complexity: $O(b^m)$
- Space Complexity: O(bm)

Optimality: No

The diagram is a schematic representation of the graph with vertices
V={A, B, C, D, E, F, G, H, I, J, K, L, M}, E={{A, B}, {A, C}, {B, D}, {B, E}, {C, F}, {C, G}, {D, H}, {D, I}, {E, J}, {E, K}, {F, L}, {F, M}}

DFS Implementation

Creating the function that takes in edges of the GRAPH which outputs the adjacency list for undirected graph.

```
from collections import defaultdict

# Generate adjacency list for undirected graph
def generateAdjacencyLst(edges):
    adjacencyList = defaultdict(list)
    for u, v in edges:
        adjacencyList[u].append(v)
        adjacencyList[v].append(u)
    return adjacencyList

edges = [['A', 'B'], ['A', 'C'], ['B', 'D'], ['B', 'E'],
         ['C', 'F'], ['C', 'G'], ['D', 'H'], ['D', 'I'],
         ['E', 'J'], ['E', 'K'], ['F', 'L'], ['F', 'M']]
adjacencyList = generateAdjacencyLst(edges)
adjacencyList
```

```
defaultdict(list,
            {'A': ['B', 'C'],
             'B': ['A', 'D', 'E'],
             'C': ['A', 'F', 'G'],
             'D': ['B', 'H', 'I'],
             'E': ['B', 'J', 'K'],
             'F': ['C', 'L', 'M'],
             'G': ['C'],
             'H': ['D'],
             'I': ['D'],
             'J': ['E'],
             'K': ['E'],
             'L': ['F'],
             'M': ['F']})
```

Creating the function that takes in the adjacency list and starting vertex which output the sequence of DFS search.

```python
# find dfs traversal from starting vertex
def dfs(adjacencyList, vertex, visitedSet = None, path = None):
    # create memo once in top-level call
    if visitedSet is None:
        visitedSet = set()
    if path is None:
        path = []

    visitedSet.add(vertex)
    path.append(vertex)
    if vertex in adjacencyList:
        for neighbor in adjacencyList[vertex]:
            if neighbor not in visitedSet:
                dfs(adjacencyList, neighbor, visitedSet, path)
    return path

print(dfs(adjacencyList, 'A'))
```

```
['A', 'B', 'D', 'H', 'I', 'E', 'J', 'K', 'C', 'F', 'L', 'M', 'G']
```

Lab Exercises:

1. Implement search algorithm techniques BFS and DFS as discussed in the example.

Conclusion:

Decision Tree

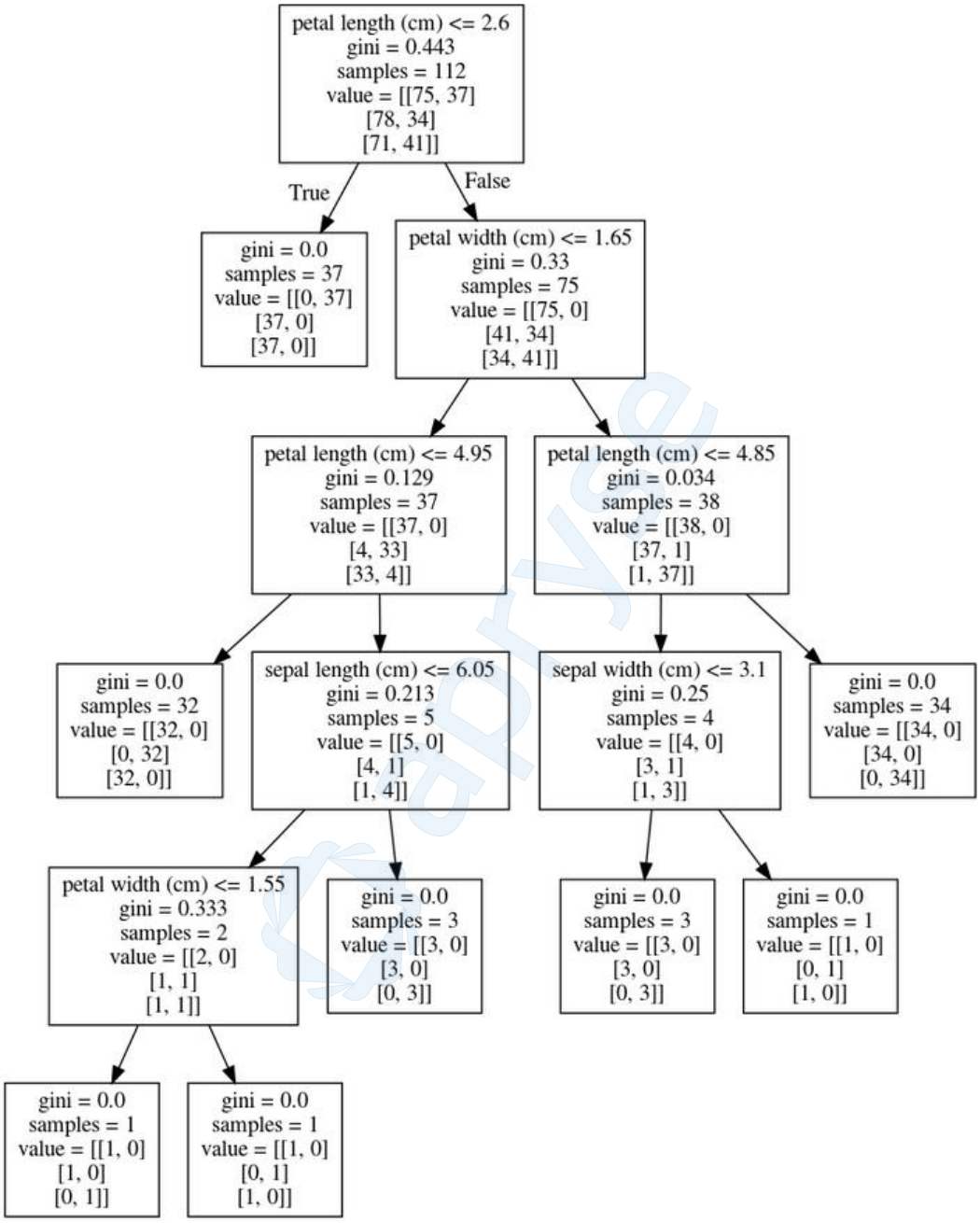
Objective:

To perform decision tree algorithm techniques in AI and observe results of the algorithm.

Background and Procedure:

Decision Tree

A Decision Tree is just a Flow Chart, and can help you make decisions based on previous experience.


petal length (cm) <= 2.6
gini = 0.443
samples = 112
value = [[75, 37]
[78, 34]
[71, 41]]
True
False
gini = 0.0
samples = 37
value = [[0, 37]
[37, 0]
[37, 0]]
petal width (cm) <= 1.65
gini = 0.33
samples = 75
value = [[75, 0]
[41, 34]
[34, 41]]
petal length (cm) <= 4.95
gini = 0.129
samples = 37
value = [[37, 0]
[4, 33]
[33, 4]]
petal length (cm) <= 4.85
gini = 0.034
samples = 38
value = [[38, 0]
[37, 1]
[1, 37]]
gini = 0.0
samples = 32
value = [[32, 0]
[0, 32]
[32, 0]]
sepal length (cm) <= 6.05
gini = 0.213
samples = 5
value = [[5, 0]
[4, 1]
[1, 4]]
sepal width (cm) <= 3.1
gini = 0.25
samples = 4
value = [[4, 0]
[3, 1]
[1, 3]]
gini = 0.0
samples = 34
value = [[34, 0]
[34, 0]
[0, 34]]
petal width (cm) <= 1.55
gini = 0.333
samples = 2
value = [[2, 0]
[1, 1]
[1, 1]]
gini = 0.0
samples = 3
value = [[3, 0]
[3, 0]
[0, 3]]
gini = 0.0
samples = 3
value = [[3, 0]
[3, 0]
[0, 3]]
gini = 0.0
samples = 1
value = [[1, 0]
[0, 1]
[1, 0]]
gini = 0.0
samples = 1
value = [[1, 0]
[1, 0]
[0, 1]]
gini = 0.0
samples = 1
value = [[1, 0]
[0, 1]
[1, 0]]

In [1]:
```
import pandas as pd
```

In [2]:
```
df = pd.read_csv("salaries.csv")
df.head()
```

Out[2]:

	company	job	degree	salary_more_then_100k
0	google	sales executive	bachelors	0
1	google	sales executive	masters	0
2	google	business manager	bachelors	1
3	google	business manager	masters	1
4	google	computer programmer	bachelors	0

In [3]:
```
inputs = df.drop('salary_more_then_100k',axis='columns')
```

In [4]:
```
target = df['salary_more_then_100k']
```

In [5]:
```
from sklearn.preprocessing import LabelEncoder
le_company = LabelEncoder()
le_job = LabelEncoder()
le_degree = LabelEncoder()
```

In [6]:
```
inputs['company_n'] = le_company.fit_transform(inputs['company'])
inputs['job_n'] = le_job.fit_transform(inputs['job'])
inputs['degree_n'] = le_degree.fit_transform(inputs['degree'])
```

In [7]:
```
inputs
```

Out[7]:

	company	job	degree	company_n	job_n	degree_n
0	google	sales executive	bachelors	2	2	0
1	google	sales executive	masters	2	2	1
2	google	business manager	bachelors	2	0	0
3	google	business manager	masters	2	0	1
4	google	computer programmer	bachelors	2	1	0
5	google	computer programmer	masters	2	1	1
6	abc pharma	sales executive	masters	0	2	1
7	abc pharma	computer programmer	bachelors	0	1	0
8	abc pharma	business manager	bachelors	0	0	0
9	abc pharma	business manager	masters	0	0	1
10	facebook	sales executive	bachelors	1	2	0
11	facebook	sales executive	masters	1	2	1
12	facebook	business manager	bachelors	1	0	0
13	facebook	business manager	masters	1	0	1
14	facebook	computer programmer	bachelors	1	1	0
15	facebook	computer programmer	masters	1	1	1

In [8]:
```
inputs_n = inputs.drop(['company','job','degree'],axis='columns')
```

In [9]:
```
inputs_n
```

Out[9]:

	company_n	job_n	degree_n
0	2	2	0
1	2	2	1
2	2	0	0
3	2	0	1
4	2	1	0
5	2	1	1
6	0	2	1
7	0	1	0
8	0	0	0
9	0	0	1
10	1	2	0
11	1	2	1
12	1	0	0
13	1	0	1
14	1	1	0
15	1	1	1

In [10]:
```
target
```

Out[10]:
```
0     0
1     0
2     1
3     1
4     0
5     1
6     0
7     0
8     0
9     1
10    1
11    1
12    1
13    1
14    1
15    1
Name: salary_more_then_100k, dtype: int64
```

In [11]:
```
from sklearn import tree
model = tree.DecisionTreeClassifier()
```

In [12]:
```
model.fit(inputs_n, target)
```

Out[12]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

In [13]:
```
model.score(inputs_n,target)
```

Out[13]:
```
1.0
```

```
In [13]: model.score(inputs_n,target)
Out[13]: 1.0
```

Is salary of Computer Engineer, Bachelors degree > 100 k ?

```
In [14]: model.predict([[2,1,0]])
Out[14]: array([0], dtype=int64)
```

Is salary of Computer Engineer, Masters degree > 100 k ?

```
In [16]: model.predict([[2,1,1]])
Out[16]: array([1], dtype=int64)
```

```
In [ ]:
```

Lab Exercises:

2. Implement decision tree algorithm on salaries data.

Conclusion: