

FACULTY OF ENGINEERING AND COMPUTER SCIENCE

NATIONAL UNIVERSITY OF MODERN LANGUAGES | ISLAMABAD

LAB MANUAL

Artificial intelligence (course code)

Artificial Intelligence (COURSE CODE)

**OBJECTIVES**

* To acquaint students with better understanding of Artificial Intelligence
* To familiarize students with Python Programming
* To equip students with hands-on experience of implementing Artificial Intelligence algorithms using Python Programming

**INTRODUCTION**

This lab is about artificial intelligence (AI), which is one of the most advanced and complicated yet advantageous emerging technologies. In this lab, students will learn about AI algorithms from basic AI agents, graph search to machine learning and deep learning using Python programming language.

**DESIGN SKILLS / TECHNIQUES PRACTICED**

Place some content related to design skills / techniques practiced in your course and lab here.

**SOFTWARE TOOLS / TECHNOLOGY INVOLVED**

* Python IDE

**EFFECTIVENESS**

This lab provides hands-on experience of implementing AI algorithms, which ultimately helps students to conduct research and develop AI-based projects.

Artificial Intelligence (COURSE CODE) 

DEPARTMENT OF SOFTWARE ENGINEERING

NATIONAL UNIVERSITY OF MODERN LANGUAGES ISLAMABAD

**LAB OUTLINE**

|  |  |
| --- | --- |
| **Week No.** | **Title of Lab** |
| 1 | Introduction to Artificial intelligence and Python and Installation of Python IDE |
| 2 | Python programming (Syntax, printing, data types and variables, conditional loops) |
| 3 | Python programming (loops, functions, classes) |
| 4 | Python programming (lists, tuples, strings, dictionaries) |
| 5 | Intelligent Agents |
| 6 | Graph Search: Uninformed search and Informed search |
| 7 | Introduction to NumPy, Pandas, Scikit-learn and Matplotlib Python Packages |
| 8 | Introduction to Machine Learning, Deep learning and deep learning Frameworks (TensorFlow, Keras) in Python |
| 9 | Supervised machine Learning: Classification with K-Nearest Neighbors |
| 10 | Supervised machine Learning: Regression with K-Nearest Neighbors |
| 11 | Unsupervised machine learning: K-mean clustering |
| 12 | Implementation of Neural Networks (NN) in Python |
| 13 | Evaluation Metrics to evaluate machine learning algorithms |
| 14 | Fuzzy Logic Systems |
| 15 | Natural Language Processing (NLP) |
| 16 | Reinforcement Learning |

[LAB 1: Introduction to Artificial intelligence and Python and Installation of Python IDE 1](#_Toc64381470)

[1.1 Objectives: 1](#_Toc64381471)

[1.2 Basic Concept of Artificial Intelligence (AI) 1](#_Toc64381472)

[1.3 The Necessity of Learning AI 2](#_Toc64381473)

[1.4 Why Python for AI 3](#_Toc64381474)

[1.5 Python and PyCharm Installation: 4](#_Toc64381475)

[LAB 2: Python programming (Syntax, printing, data types and variables, conditional loops) 5](#_Toc64381476)

[2.1 Objectives: 5](#_Toc64381477)

[2.2 Learning python for a C++/C# programmer 5](#_Toc64381478)

[2.3 Math in Python 5](#_Toc64381479)

[2.4 Python Operators 6](#_Toc64381480)

[2.5 Comments in Python: 6](#_Toc64381481)

[2.6 Variables: 6](#_Toc64381482)

[2.7 Relational operators: 7](#_Toc64381483)

[2.8 Boolean Logic: 7](#_Toc64381484)

[2.9 Operator Precedence: 7](#_Toc64381485)

[2.10 Conditional Statements: 8](#_Toc64381486)

[2.11 Input from user: 9](#_Toc64381487)

[2.12 The print() function 9](#_Toc64381488)

[2.13 The print() function - instructions 9](#_Toc64381489)

[2.14 TASKS: 10](#_Toc64381490)

[LAB 3: Python programming (loops, functions, classes) 11](#_Toc64381491)

[3.1 Objectives: 11](#_Toc64381492)

[3.2 Loops in Python: 11](#_Toc64381493)

[3.3 Functions: 11](#_Toc64381494)

[3.4 Classes & Inheritance: 13](#_Toc64381495)

[3.5 Module: 15](#_Toc64381496)

[3.6 TASKS: 15](#_Toc64381497)

[LAB 4: Python programming (lists, tuples, strings, dictionaries) 17](#_Toc64381498)

[4.1 Objectives: 17](#_Toc64381499)

[4.2 Strings: 17](#_Toc64381500)

[4.2 Lists (Mutable): 18](#_Toc64381501)

[4.3 Tuples (imutable): 21](#_Toc64381502)

[4.4 Sets: 21](#_Toc64381503)

[4.5 Dictionaries: 23](#_Toc64381504)

[4.6 TASKS: 24](#_Toc64381505)

[LAB 5: Intelligent Agents 25](#_Toc64381506)

[5.1 Objectives: 25](#_Toc64381507)

[5.2 Agents and Environment 25](#_Toc64381508)

[5.3 Types of Agents 26](#_Toc64381509)

[5.4 Simple Reflex Agent 26](#_Toc64381510)

[5.5 Model Reflex Agent 27](#_Toc64381511)

[LAB 6: Graph Search: Uninformed search and Informed search 32](#_Toc64381512)

[6.1 Objectives: 32](#_Toc64381513)

[6.2 Depth-first search: 32](#_Toc64381514)

[6.3 Breadth-first search: 34](#_Toc64381515)

[6.4 TASKS 36](#_Toc64381516)

[LAB 7: Introduction to NumPy, Pandas, Scikit-learn and Matplotlib Python Packages 38](#_Toc64381517)

[7.1 Objectives: 38](#_Toc64381518)

[7.2 Different Python Packages 38](#_Toc64381519)

[7.2.1 NUMPY 38](#_Toc64381520)

[7.2.2 SCIKIT-Learn 39](#_Toc64381521)

[7.2.3 PANDAS 39](#_Toc64381522)

[7.2.4 Matplotlib 40](#_Toc64381523)

[7.3 TASKS 41](#_Toc64381524)

[LAB 8: Introduction to Machine Learning, Deep learning and deep learning Frameworks (TensorFlow, Keras) in Python 42](#_Toc64381525)

[8.1 Objectives: 42](#_Toc64381526)

[8.2 Machine Learning and deep learning 42](#_Toc64381527)

[8.2.1 Learning from experience 42](#_Toc64381528)

[8.2.2 Machine learning tasks 43](#_Toc64381529)

[8.2.3 Training data, testing data, and validation data 43](#_Toc64381530)

[8.2.4 Deep Learning 44](#_Toc64381531)

[8.2.5 What is TensorFlow? 45](#_Toc64381532)

[8.2.6 What is Keras? 45](#_Toc64381533)

[8.2.7 How to create deep learning environment? 45](#_Toc64381534)

[8.3 TASK: 46](#_Toc64381535)

[LAB 9: Supervised machine Learning: Classification with K-Nearest Neighbors 47](#_Toc64381536)

[9.1 Objectives: 47](#_Toc64381537)

[9.2 K-Nearest Neighbors 47](#_Toc64381538)

[9.2.1 Implementation of K-NN Classifier 48](#_Toc64381539)

[9.3 TASK: 49](#_Toc64381540)

[LAB 10: Supervised machine Learning: Regression with K-Nearest Neighbors 50](#_Toc64381541)

[10.1 Objectives: 50](#_Toc64381542)

[10.2 K-Nearest Neighbors 50](#_Toc64381543)

[10.2.1 Implementation of K-NN Regressor 51](#_Toc64381544)

[10.3 TASK: 52](#_Toc64381545)

[LAB 11: Unsupervised machine learning: K-mean clustering 53](#_Toc64381546)

[11.1 Objectives: 53](#_Toc64381547)

[11.2 Unsupervised machine learning 53](#_Toc64381548)

[11.2.1 Clustering 53](#_Toc64381549)

[11.2.2 K-means 54](#_Toc64381550)

[11.3 TASK: 55](#_Toc64381551)

[LAB 12: Implementation of Neural Networks (NN) in Python 56](#_Toc64381552)

[12.1 Objectives: 56](#_Toc64381553)

[12.2 Artificial Neural Networks 56](#_Toc64381554)

[12.2.1 Backpropagation 56](#_Toc64381555)

[12.2.2 Multilayer Perceptrons 57](#_Toc64381556)

[12.2.3 Activation function 57](#_Toc64381557)

[12.2.4 Loss function 57](#_Toc64381558)

[12.2.5 Optimizers 57](#_Toc64381559)

[12.2.6 Implementation: 57](#_Toc64381560)

[12.3 TASK: 58](#_Toc64381561)

[LAB 13: Evaluation Metrics to evaluate machine learning algorithms 59](#_Toc64381562)

[12.1 Objectives: 59](#_Toc64381563)

[12.2 Confusion Matrix 59](#_Toc64381564)

[12.2.1 Accuracy 60](#_Toc64381565)

[12.2.2 Area Under Curve 60](#_Toc64381566)

[12.2.3 F1 Score 61](#_Toc64381567)

[12.2.4 Mean Absolute Error 62](#_Toc64381568)

[12.2.5 Mean Squared Error 62](#_Toc64381569)

[12.3 Python code 62](#_Toc64381570)

# **LAB 1: Introduction to Artificial intelligence and Python and Installation of Python IDE**

# Objectives:

* An introduction to Artificial intelligence and Python programming language

Since the invention of computers or machines, their capability to perform various tasks has experienced an exponential growth. Humans have developed the power of computer systems in terms of their diverse working domains, their increasing speed, and reducing size with respect to time.

A branch of Computer Science named Artificial Intelligence pursues creating the computers or machines as intelligent as human beings.

# Basic Concept of Artificial Intelligence (AI)

According to the father of Artificial Intelligence, John McCarthy, it is “The science and engineering of making intelligent machines, especially intelligent computer programs”.

Artificial Intelligence is a way of making a computer, a computer-controlled robot, or a software think intelligently, in the similar manner the intelligent humans think. AI is accomplished by studying how human brain thinks and how humans learn, decide, and work while trying to solve a problem, and then using the outcomes of this study as a basis of developing intelligent software and systems.

While exploiting the power of the computer systems, the curiosity of human, lead him to wonder, “Can a machine think and behave like humans do?”

Thus, the development of AI started with the intention of creating similar intelligence in machines that we find and regard high in humans.

# The Necessity of Learning AI

As we know that AI pursues creating the machines as intelligent as human beings. There are numerous reasons for us to study AI. The reasons are as follows:

## AI can learn through data

In our daily life, we deal with huge amount of data and human brain cannot keep track of so much data. That is why we need to automate the things. For doing automation, we need to study AI because it can learn from data and can do the repetitive tasks with accuracy and without tiredness.

## AI can teach itself

It is very necessary that a system should teach itself because the data itself keeps changing and the knowledge which is derived from such data must be updated constantly. We can use AI to fulfill this purpose because an AI enabled system can teach itself.

## AI can respond in real time

Artificial intelligence with the help of neural networks can analyze the data more deeply. Due to this capability, AI can think and respond to the situations which are based on the conditions in real time.

## AI achieves accuracy

With the help of deep neural networks, AI can achieve tremendous accuracy. AI helps in the field of medicine to diagnose diseases such as cancer from the MRIs of patients.

## AI can organize data to get most out of it

The data is an intellectual property for the systems which are using self-learning algorithms. We need AI to index and organize the data in a way that it always gives the best results.

# Why Python for AI

Artificial intelligence is considered to be the trending technology of the future. Already there are a number of applications made on it. Due to this, many companies and researchers are taking interest in it. But the main question that arises here is that in which programming language can these AI applications be developed? There are various programming languages like Lisp, Prolog, C++, Java and Python, which can be used for developing applications of AI. Among them, Python programming language gains a huge popularity and the reasons are as follows:

## Simple syntax & less coding

Python involves very less coding and simple syntax among other programming languages which can be used for developing AI applications. Due to this feature, the testing can be easier and we can focus more on programming.

## Inbuilt libraries for AI projects

A major advantage for using Python for AI is that it comes with inbuilt libraries. Python has libraries for almost all kinds of AI projects. For example, NumPy, SciPy, matplotlib, nltk, SimpleAI are some the important inbuilt libraries of Python.

## Open source:Python is an open source programming language. This makes it widely popular in the community.

## Can be used for broad range of programming: Python can be used for a broad range of programming tasks like small shell script to enterprise web applications. This is another reason Python is suitable for AI projects.

## Easy-to-learn − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.

## Easy-to-read − Python code is more clearly defined and visible to the eyes.

## Easy-to-maintain − Python's source code is fairly easy-to-maintain.

## A broad standard library− Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

# Python and PyCharm Installation:

## Download python interpreter

[**https://www.python.org/**](https://www.python.org/) **(Latest version** Python 3.x.x)

## Pycharm

<https://www.guru99.com/how-to-install-python.html>

# **LAB 2: Python programming (Syntax, printing, data types and variables, conditional loops)**

# Objectives:

* To get familiar with the syntax of Python
* To get familiar with data types and variables in Python
* To get familiar with conditional loops in Python
* To get familiar with the print function in Python

# Learning python for a C++/C# programmer

Let us try to quickly compare the syntax of python with that of C++/C#:

|  |  |  |
| --- | --- | --- |
|  | **C++/C#** | **Python** |
| Comment begins with | // | # |
| Statement ends with | ; | No semi-colon needed |
| Blocks of code | Defined by {} | Defined by indentation (usually four spaces) |
| Indentation of code and use of white space | Is irrelevant | Must be same for same block of code (for example for a set of statements to be executed after a particular if statement) |
| Conditional statement | if-else if- else | if – elif – else: |
| Parentheses for loop execution condition | Required | Not required but loop condition followed by a colon :  while a < n:  print(a) |

# Math in Python

Calculations are simple with Python, and expression syntax is straightforward: the operators +, -, \* and / work as expected; parentheses () can be used for grouping.

# Python 3: Simple arithmetic

>>> 1 / 2

0.5

>>> 2 \*\* 3 #Exponent operator

8

>>> 17 / 3 # classic division returns a float

5.666666666666667

>>> 17 // 3 # floor division

5

>>> 23%3 #Modulus operator

2

# Python Operators

|  |  |  |  |
| --- | --- | --- | --- |
| **Command** | **Name** | **Example** | **Output** |
| + | Addition | 4+5 | 9 |
| - | Subtraction | 8-5 | 3 |
| \* | Multiplication | 4\*5 | 20 |
| / | Classic Division | 19/3 | 6.3333 |
| % | Modulus | 19%3 | 5 |
| \*\* | Exponent | 2\*\*4 | 16 |
| // | Floor Division | 19/3 | 6 |

# Comments in Python:

#I am a comment. I can say whatever I want!

# Variables:

print ("This program is a demo of variables")

v = 1

print ("The value of v is now", v)

v = v + 1

print ("v now equals itself plus one, making it worth", v)

print ("To make v five times bigger, you would have to type v = v \* 5")

v = v \* 5

print ("There you go, now v equals", v, "and not", v / 5 )

## Strings:

word1 = "Good"

word2 = "Morning"

word3 = "to you too!"

print (word1, word2)

sentence = word1 + " " + word2 + " " +word3

print (sentence)

# Relational operators:

|  |  |
| --- | --- |
| Expression | Function |
| < | less than |
| <= | less than or equal to |
| > | greater than |
| >= | greater than or equal to |
| != | not equal to |
| == | is equal to |

# Boolean Logic:

Boolean logic is used to make more complicated conditions for **if** statements that rely on more than one condition. Python’s Boolean operators are **and**, **or**, and **not**. The **and** operator takes two arguments, and evaluates as **True** if, and only if, both of its arguments are True. Otherwise it evaluates to **False**. The **or** operator also takes two arguments. It evaluates if either (or both) of its arguments are **False**. Unlike the other operators we’ve seen so far, **not** only takes one argument and inverts it. The result of **not True** is **False**, and **not False** is **True**.

# Operator Precedence:

|  |  |
| --- | --- |
| **Operator** | **Description** |
| () | Parentheses |
| \*\* | Exponentiation (raise to the power) |
| ~ + - | Complement, unary plus and minus |
| \* / % // | 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 |

# Conditional Statements:

**‘if' - Statement**

y = 1

if y == 1:

print ("y still equals 1, I was just checking")

**‘if - else' - Statement**

a = 1

if a > 5:

print ("This shouldn't happen.")

else:

print ("This should happen.")

**‘elif' - Statement**

z = 4

if z > 70:

print ("Something is very wrong")

elif z < 7:

print ("This is normal")

# Input from user:

The **input()** function prompts for input and returns a string.

a = input (“Enter Value for variable a: ”)

print (a)

# The print() function

Three important questions have to be answered as soon as possible:

1. What is the effect the print() function causes?

The effect is very useful and very spectacular. The function:

• takes its arguments (it may accept more than one argument and may also accept less than one argument)

• converts them into human-readable form if needed (as you may suspect, strings don't require this action, as the string is already readable)

• and sends the resulting data to the output device (usually the console); in other words, anything you put into the print() function will appear on your screen.

No wonder then, that from now on, you'll utilize print() very intensively to see the results of your operations and evaluations.

2. What arguments does print() expect?

Any. We'll show you soon that print() is able to operate with virtually all types of data offered by Python. Strings, numbers, characters, logical values, objects - any of these may be successfully passed to print().

3. What value does the print() function evaluate?

None. Its effect is enough - print() does not evaluate anything.

# The print() function - instructions

You already know that this program contains one function invocation. In turn, the function invocation is one of the possible kinds of Python instruction. Ergo, this program consists of just one instruction.

Of course, any complex program usually contains many more instructions than one. The question is: how do you couple more than one instruction into the Python code?

Python's syntax is quite specific in this area. Unlike most programming languages, Python requires that there cannot be more than one instruction in a line.

A line can be empty (i.e., it may contain no instruction at all) but it must not contain two, three or more instructions. This is strictly prohibited.

Note: Python makes one exception to this rule - it allows one instruction to spread across more than one line (which may be helpful when your code contains complex constructions).

# TASKS:

## TASK 1:

Write a program that first displays a simple cafe menu (see example below), asks the user to enter the number of a choice, and either prints the appropriate action OR prints an error message that their choice was not valid.

Example output:

1. Soup and salad

2. Pasta with meat sauce

3. Chef's special

Which number would you like to order? 2 One Pasta with meat sauce coming right up!

Another example output:

1. Soup and salad   
2. Pasta with meat sauce   
3. Chef's special

Which number would you like to order? 5

Sorry, that is not a valid choice.

## TASK 2:

Once upon a time in Apple land, John had three apples, Mary had five apples, and Adam had six apples. They were all very happy and lived for a long time. End of story.

Your task is to:

• create the variables: john, mary, and adam;

• assign values to the variables. The values must be equal to the numbers of fruit possessed by John, Mary, and Adam respectively;

• having stored the numbers in the variables, print the variables on one line, and separate each of them with a comma;

• now create a new variable named totalApples equal to addition of the three former variables.

• print the value stored in totalApples to the console

• Check if the totalApples is greater, smaller or equal to 10

# **LAB 3: Python programming (loops, functions, classes)**

# Objectives:

* To learn and implement loops, functions and classes

# Loops in Python:

## The 'while' loop

a = 0

while a < 10:

a = a + 1

print (a )

## Range function:

Range(5) #[0,1,2,3,4]

Range(1,5) #[1,2,3,4]

Range(1,10,3) #[1,4,7]

## The 'for' loop

for i in range(1, 5):

print (i )

for i in range(1, 5):

print (i)

else:

print ('The for loop is over')

# Functions:

## How to call a function?

function\_name(parameters)

Code Example - Using a function

def greet(): #function definition

print(“Hello”)

print(“Good Morning”)

greet() #function calling

def add\_sub(x,y)

a=x+y

b=x-y

return a,b

result1, result2 = add\_sub(5,10)

print(result1, result2)

def multiplybytwo(x):

return x\*2

a = multiplybytwo(70)

The computer would actually see this:

a=140

## Define a Function?

def function\_name(parameter\_1,parameter\_2):

{this is the code in the function}

return {value (e.g. text or number) to return to the main program}

## range() Function:

If you need to iterate over a sequence of numbers, the built-in function range() comes in handy. It generates iterator containing arithmetic progressions:

>>> range(10) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

It is possible to let the range start at another number, or to specify a different increment (even negative; sometimes this is called the ‘step’):

>>> list(range(5, 10))

[5, 6, 7, 8, 9]

>>> list(range(0, 10, 3) )

[0, 3, 6, 9]

>>> list(range(-10, -100, -30) )

[-10, -40, -70]

The range() function is especially useful in loops.

# Classes & Inheritance:

The word 'class' can be used when describing the code where the class is defined.

A variable inside a class is known as an *Attribute*

A function inside a class is known as a *method*

* A class is like a
  + Prototype
  + Blue-print
  + An object creator
* A class defines potential objects
  + What their structure will be
  + What they will be able to do
* Objects are instances of a class
  + An object is a container of data: attributes
  + An object has associated functions: methods

## Syntax:

# Defining a class

class class\_name:

[statement 1]

[statement 2]

[statement 3] [etc]

## Inheritance Syntax:

class child\_class(parent\_class):

def \_\_init\_\_(self,x):

# it will modify the \_init\_ function from parent class

# additional methods can be defined here

**‘self’ keyword:**

The first argument of every class method, including \_\_init\_\_, is always a reference to the current instance of the class. By convention, this argument is always named self. In the \_\_init\_\_ method, self refers to the newly created object; in other class methods, it refers to the instance whose method was called.

Anytime you create an object of the class, the first thing it does before going on to any other line is it looks for init function and it calls whatever is written in here. You don’t have to call it explicitly like any other function

**Example1:**

**class MyClass:**

**i = 12345**

**def f(self):**

**return 'hello world'**

**x = MyClass()**

**print (x.i)**

**print (x.f())**

**Example2:**

class Complex:

def \_\_init\_\_(self, realpart, imagpart):

self.r = realpart

self.i = imagpart

x = Complex(3.0, -4.5)

print (x.r," ",x.i )

**Example3:**

**class Shape:**

**def \_\_init\_\_(self,x,y): #The \_\_init\_\_ function always runs first**

**self.x = x**

**self.y = y**

**description = "This shape has not been described yet"**

**author = "Nobody has claimed to make this shape yet"**

**def area(self):**

**return self.x \* self.y**

**def perimeter(self):**

**return 2 \* self.x + 2 \* self.y**

**def describe(self,text):**

**self.description = text**

**def authorName(self,text):**

**self.author = text**

**def scaleSize(self,scale):**

**self.x = self.x \* scale**

**self.y = self.y \* scale**

**a=Shape(3,4)**

**print (a.area())**

**Inheritance Example:**

class Square(Shape):

def \_\_init\_\_(self,x):

self.x = x

self.y = x

class DoubleSquare(Square):

def \_\_init\_\_(self,y):

self.x = 2 \* y

self.y = y

def perimeter(self):

return 2 \* self.x + 2 \* self.y

# Module:

A module is a python file that (generally) has only definitions of variables, functions, and classes.

**Example:** Module name mymodule.py

# Define some variables:

ageofqueen = 78

# define some functions

def printhello():

print ("hello")

# define a class

class Piano:

def \_\_init\_\_(self):

self.type = input("What type of piano?: ")

self.height = input("What height (in feet)?: ")

self.price = input("How much did it cost?: ")

self.age = input("How old is it (in years)?: ")

def printdetails(self):

print ("This piano is a/an " + self.height + " foot")

print (self.type, "piano, " + self.age, "years old and costing " + self.price + " dollars.")

## Importing module in main program:

### mainprogam.py ##

# IMPORTS ANOTHER MODULE

import mymodule

print (mymodule.ageofqueen )

cfcpiano = mymodule.Piano()

cfcpiano.printdetails()

Another way of importing the module is:

from mymodule import Piano, ageofqueen

print (ageofqueen)

cfcpiano = Piano()

cfcpiano.printdetails()

# TASKS:

## LAB TASK 1:

Write a program to find the largest of ten numbers provided by user, using functions.

## LAB TASK 2:

Create a class name basic\_calc with following attributes and methods;

Two integers (values are passed with instance creation)

Different methods such as addition, subtraction, division, multiplication

# **LAB 4: Python programming (lists, tuples, strings, dictionaries)**

# Objectives:

* To get familiar with strings, lists, tuples and dictionaries in Python

# Strings:

## Indexes of String:

Characters in a string are numbered with *indexes* starting at 0:

Example:

name = "J. Smith”

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | 00 | 11 | 22 | 33 | 44 | 55 | 66 | 77 |
| Character | pJ | .. |  | AS | im | Si | Ht | ah |

Accessing an individual character of a string:

**variableName** [ **index** ]

Example:

print (name, " starts with", name[0])

Output:

J. Smith starts with J

## input:

input: Reads a string of text from user input.

Example:

name = input("What's your name? ")

print (name, "... what a nice name!")

Output:

What's your name? Ali

Ali... what a nice name!

## String Properties:

len(*string*) - number of characters in a string (including spaces)

str.lower(*string*) - lowercase version of a string

str.upper(*string*) - uppercase version of a string

Example:

name = "Linkin Park"

length = len(name)

big\_name = str.upper(name)

print (big\_name, "has", length, "characters")

Output:

LINKIN PARK has 11 characters

## Strings and numbers:

ord(*text*) - converts a string into a number.

Example: ord(‘a’) is 97, ord("b") is 98, ...

Characters map to numbers using standardized mappings such as *ASCII* and *Unicode*.

chr (*number*) - converts a number into a string.

Example: chr(99) is "c"

# Lists (Mutable):

Most of our variables have one value in them - when we put a new value in the variable, the old value is overwritten

x = 2

x = 4

print(x)

4

A collection allows us to put many values in a single “variable”

A collection is nice because we can carry all many values around in one convenient package.

Strings are “immutable” - we cannot change the contents of a string - we must make a new string to make any change

Lists are “mutable” - we can change an element of a list using the index operator

Lists are what they seem - a list of values. Each one of them is numbered, starting from zero. You can remove values from the list, and add new values to the end. Example: Your many cats' names. *Compound* data types, used to group together other values. The most versatile is the *list*, which can be written as a list of comma-separated values (items) between square brackets. List items need not all have the same type.

>>> num = [1,2,3]

>>> names = ['Talal', 'Husnain', 'Saeed', 'Aezid']

>>> hybrid = [5,5.6,'text']

>>> combined = [num,names,hybrid]

>>> combined

[[1, 2, 3], ['Talal', 'Husnain', 'Saeed', 'Aezid'], [5, 5.6, 'text']]

>>>

cats = ['Tom', 'Snappy', 'Kitty', 'Jessie', 'Chester']

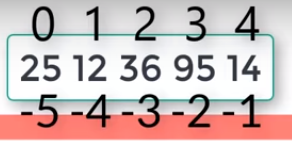
print (cats[2])

cats.append(‘Oscar’)

print (len(cats))

#Remove 2nd cat, Snappy.

del cats[1]



## Compound datatype:

>>> a = ['spam', 'eggs', 100, 1234]

A[:3]

A[3:]

>>> a[1:-1] #start at element at index 1, end before last element

['eggs', 100]

>>> a[:2] + ['bacon', 2\*2]

['spam', 'eggs', 'bacon', 4]

>>> 3\*a[:3] + ['Boo!']

['spam', 'eggs', 100, 'spam', 'eggs', 100, 'spam', 'eggs', 100, 'Boo!']

>>> a= ['spam', 'eggs', 100, 1234]

>>> a[2] = a[2] + 23

>>> a

['spam', 'eggs', 123, 1234]

## Replace some items:

>>> a[0:2] = [1, 12]

>>> a

[1, 12, 123, 1234]

## Remove some:

>>> a[0:2] = []

>>> a

[123, 1234]

## Clear the list: replace all items with an empty list:

>>> a[:] = []

>>> a

[]

## Length of list:

>>> a = ['a', 'b', 'c', 'd']

>>> len(a)

4

## Nested lists:

>>> q = [2, 3]

>>> p = [1, q, 4]

>>> len(p)

3

>>> p[1]

[2, 3].

Del nums [3:]

This is used to remove multiple values and this will remove from values after index number 3

## Functions of lists:

**list.append(x):** Add an item to the end of the list; equivalent to a[len(a):] = [x].

**list.extend(L):** Extend the list by appending all the items in the given list; equivalent to a[len(a):] = L.

**list.insert(i, x):** Insert an item at a given position. The first argument is the index of the element before which to insert, so a.insert(0, x) inserts at the front of the list.

**list.remove(x):** Remove the first item from the list whose value is x. It is an error if there is no such item. (based on number you entered)

**list.pop(i):** Remove the item at the given position in the list, and return it. If no index is specified, a.pop() removes and returns the last item in the list. (based on index number you entered)

If you don’t specify the index number the last element will be removed. Concept of stack (LIFO)

**list.count(x):** Return the number of times x appears in the list.

**list.sort():** Sort the items of the list, in place.

**list.reverse():** Reverse the elements of the list, in place.

# Tuples (imutable):

Tuples are just like lists, but you can't change their values. Again, each value is numbered starting from zero, for easy reference. Example: the names of the months of the year.

Square brackets are used for list so parenthesis are used for tuples

months = ('January' , 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November', 'December')

|  |  |
| --- | --- |
| **Index** | **Value** |
| 0 | January |
| 1 | February |
| 2 | March |
| 3 | April |
| 4 | May |
| 5 | June |
| 6 | July |
| 7 | August |
| 8 | September |
| 9 | October |
| 10 | November |
| 11 | December |

We can have easy membership tests in Tuples using the keyword in.

>>> 'December' in months # fast membership testing

True

# Sets:

A set is an unordered collection with no duplicate elements. Basic uses include membership testing and eliminating duplicate entries. Set objects also support mathematical operations like union, intersection, difference, and symmetric difference.

Curly braces or the set() function can be used to create sets. Note: to create an empty set you have to use set(), not {}; the latter creates an empty dictionary.

Example 1:

>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']

>>> fruit = set(basket) # create a set without duplicates

>>> fruit

{'banana', 'orange', 'pear', 'apple' }

>>> 'orange' in fruit # fast membership testing

True

>>> 'crabgrass' in fruit

False

Example 2:

>>> # Demonstrate set operations on unique letters from two words

>>> a = set('abracadabra')

>>> b = set('alacazam')

>>> a # unique letters in a

{'a', 'r', 'b', 'c', 'd'}

>>> a - b # letters in a but not in b

{'r', 'd', 'b'}

>>> a | b # letters in either a or b

{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}

>>> a & b # letters in both a and b

{'a', 'c'}

>>> a ^ b # letters in a or b but not both

{'r', 'd', 'b', 'm', 'z', 'l'}

Set comprehensions are also supported:

>>> a = {x for x in 'abracadabra' if x not in 'abc'}

>>> a

{'r', 'd'}

# Dictionaries:

Dictionaries are similar to what their name suggests - a dictionary. In a dictionary, you have an 'index' of words, and for each of them a definition.

In python, the word is called a 'key', and the definition a 'value'. The values in a dictionary aren't numbered - they aren't in any specific order, either - the key does the same thing.

You can add, remove, and modify the values in dictionaries. Example: telephone book.

The main operations on a dictionary are storing a value with some key and extracting the value given the key. It is also possible to delete a key:value pair with del. If you store using a key that is already in use, the old value associated with that key is forgotten. It is an error to extract a value using a non-existent key.

Performing list(d.keys()) on a dictionary returns a list of all the keys used in the dictionary, in arbitrary order (if you want it sorted, just use sorted(d.keys()) instead). To check whether a single key is in the dictionary, use the **in** keyword.

At one time, only one value may be stored against a particular key. Storing a new value for an existing key overwrites its old value. If you need to store more than one value for a particular key, it can be done by storing a list as the value for a key.

phonebook = {'ali':8806336, 'omer':6784346,'shoaib':7658344, 'saad':1122345}

#Add the person '' to the phonebook:

phonebook['waqas'] = 1234567

print("Original Phonebook")

print(phonebook)

# Remove the person 'shoaib' from the phonebook:

del phonebook['shoaib']

print("'shoaib' deleted from phonebook")

print(phonebook)

phonebook = {'Andrew Parson':8806336, \

'Emily Everett':6784346, 'Peter Power':7658344, \

'Louis Lane':1122345}

print("New phonebook")

print(phonebook)

#Add the person 'Gingerbread Man' to the phonebook:

phonebook['Gingerbread Man'] = 1234567

list(phonebook.keys())

sorted(phonebook.keys())

print( 'waqas' in phonebook)

print( 'Emily Everett' in phonebook)

#Delete the person 'Gingerbread Man' from the phonebook:

del phonebook['Gingerbread Man']

# TASKS:

## LAB TASK1:

Write Python Program to Calculate the Length of a String

Without Using Built-In len() Function.

## LAB TASK 2:

Write a program that creates a list of 10 random integers. Then create two lists by

name odd\_list and even\_list that have all odd and even values of the list respectively.

# **LAB 5: Intelligent Agents**

# Objectives:

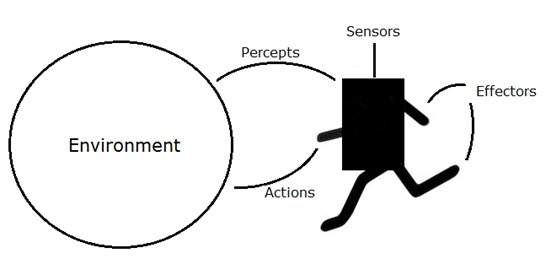
1. To learn and implement intelligent agents using Python

# Agents and Environment

An AI system is composed of an agent and its environment. The agents act in their environment. The environment may contain other agents.

An **agent** is anything that can perceive its environment through **sensors** and acts upon that environment through **effectors.**

* A **human agent** has sensory organs such as eyes, ears, nose, tongue and skin parallel to the sensors, and other organs such as hands, legs, mouth, for effectors.
* A **robotic agent** replaces cameras and infrared range finders for the sensors, and various motors and actuators for effectors.
* A **software agent** has encoded bit strings as its programs and actions.



## Agent Terminology

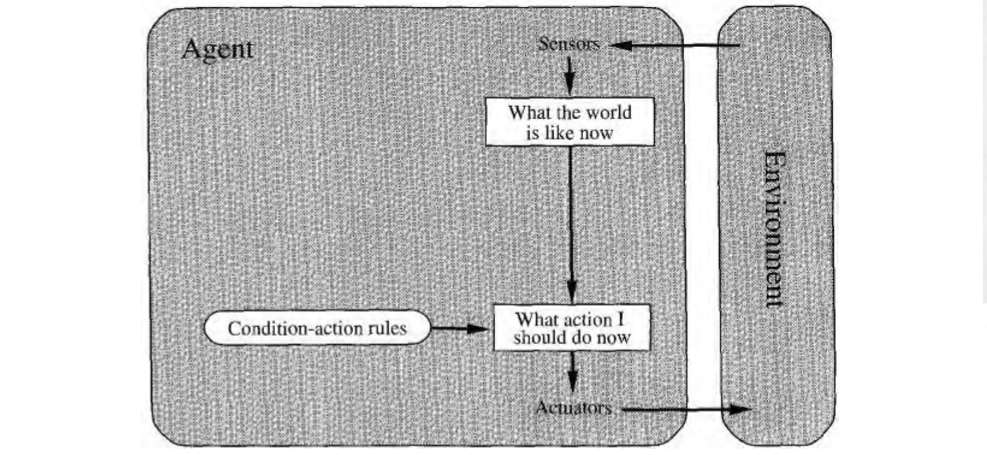
* **Performance Measure of Agent** − It is the criteria, which determines how successful an agent is.
* **Behavior of Agent** − It is the action that agent performs after any given sequence of percepts.
* **Percept** − It is agent’s perceptual inputs at a given instance.
* **Percept Sequence** − It is the history of all that an agent has perceived till date.
* **Agent Function** − It is a map from the precept sequence to an action.
* **Agent has;**
  + Sensors
  + Actuators
  + Percepts
  + Actions

# Types of Agents

1. Table-driven agent
2. Reflex agent
3. Model-based reflex agent
4. Goal-based agent
5. Utility-based agent
6. Learning agent

# Simple Reflex Agent

Simple reflex agents act only on the basis of the current percept, ignoring the rest of the percept history. The agent function is based on the condition-action rule: if condition then action. This agent function only succeeds when the environment is fully observable. Some reflex agents can also contain information on their current state which allows them to disregard conditions whose actuators are already triggered. A schematic diagram of a simple reflex agent is shown below:



* They choose actions only based on the current percept.
* They are rational only if a correct decision is made only on the basis of current precept.
* Their environment is completely observable.

# Model Reflex Agent

Model-based reflex agents are made to deal with partial accessibility; they do this by keeping track of the part of the world it can see now. It does this by keeping an internal state that depends on what it has seen before so it holds information on the unobserved aspects of the current state.

This time out mars Lander after picking up its first sample, it stores this in the internal state of the world around it so when it come across the second same sample it passes it by and saves space for other samples.

While reading this you are keeping track of where you have got to somewhere internally in your brain just in case you lose your place.

But in order to update this internal store we need 2 things:

1. Information on how the world evolves on its own.

e.g. If our mars Lander picked up the rock next to the one it was going to the world around it would carry on as normal

2. How the world is affected by the agents actions.

E.g. If our mars Lander took a sample under a precarious ledge it could displace a rock and it could be crushed.

We can predict how the world will react with facts like if you remove a supporting rock under a ledge the ledge will fall, such facts are called models, hence the name model-based agent.

* If the world is not fully observable, the agent must remember observations about the parts of the environment it cannot currently observe.
* This usually requires an internal representation of the world (or internal state).

A schematic diagram of a model based reflex agent is shown below:

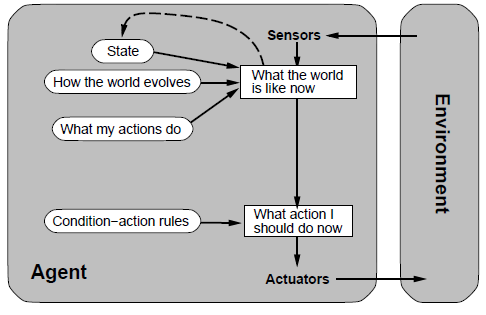


Figure 1: schematic of Model based reflex agent

They use a model of the world to choose their actions. They maintain an internal state.

**Model** − knowledge about “how the things happen in the world”.

**Internal State** − It is a representation of unobserved aspects of current state depending on percept history.

**Updating the state requires the information about −**

* How the world evolves.
* How the agent’s actions affect the world..

## Pseudocodes:



Example: Room temperature controller

sensor=[10,20,30,40,50]  
def room\_temperature(sensor):  
 for sens in sensor:  
 if sens<30:  
 print(**"fan off"**)  
 elif sens>30:  
 print(**"fan on"**)  
 return 0  
print(room\_temperature(sensor))



Example: Vacuum cleaner

Consider the vacuum world shown in the figure below:

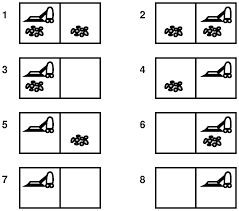


Figure 2: vacuum cleaner problem

This particular world has just two locations: squares A and B. The vacuum agent perceives which square it is in and whether there is dirt in the square. It can choose to move left, move right, suck up the dirt, or do nothing. **Agent function** is the following: if the current square is dirty, then suck, otherwise move to the other square. Write a model based reflex agent for the vacuum cleaner. (Hint: Agent has initial states knowledge)

If the current square is dirty, then suck; otherwise, move to the other square.

Initial state is 1, where square A and Square B, both are dirty.

Pseudocode to the problem is as follows;

function Reflex-Vacuum-Agent( [location,status]) returns an action

static: last A, last B, numbers, initially ∞

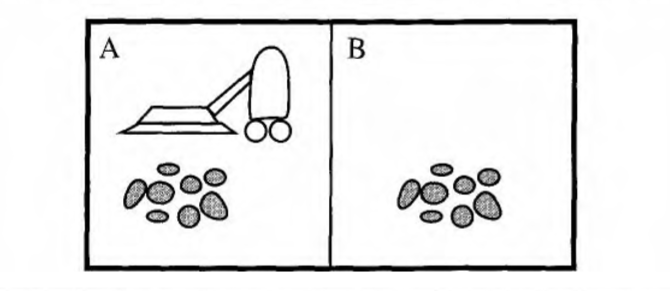
if status = Dirty then and so on

## code:

class ModelBasedVacuumAgent():  
 def \_\_init\_\_(self,init\_a,init\_b):  
 self.model = {**"Loc\_a"** : init\_a, **"Loc\_b"** : init\_b}  
 def DoAction(self,location, status):  
 self.model[location] = status  
 print(self.model)  
 if self.model[**"Loc\_a"**] == self.model[**"Loc\_b"**] == **'clean'**:  
 return **'NoOp'** elif status == **'dirty'**:  
 return **'suck'** elif location == **"Loc\_a"**:  
 return **'right'** else:  
 return **'left'**a=ModelBasedVacuumAgent(**'dirty'**,**'dirty'**)  
print(a.DoAction(**"Loc\_a"**,**'dirty'**))

## TASK:

* + Can you name few model-based reflex agents?
  + Write a program for model-based reflex agent of your own choice.
  + Consider the vacuum world shown in the figure below:



This particular world has just two locations: squares A and B. The vacuum agent perceives which square it is in and whether there is dirt in the square. It can choose to move left, move right, suck up the dirt, or do nothing. One very simple **agent function** is the following: if the current square is dirty, then suck, otherwise move to the other square. Write a simple reflex agent for the vacuum cleaner. (Hint: Agent has no initial states knowledge)

If the current square is dirty, then suck; otherwise, move to the other square.

Pseudocode to the task is as follows;

function Reflex-Vacuum-Agent( [location,status]) returns an action

if status = Dirty then return Suck

else if location = A then return Right

else if location = B then return Left

# **LAB 6: Graph Search: Uninformed search and Informed search**

# Objectives:

1. To learn and Implement Depth-First-Search algorithm

# Depth-first search:

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. One starts at the root (selecting some arbitrary node as the root in the case of a graph) and explores as far as possible along each branch before backtracking.

## Pseudocode:

Input: A graph G and a vertex v of G

Output: All vertices reachable from v labelled as discovered

A recursive implementation of DFS:

1 procedure DFS(G,v):

2 label v as discovered

3 for all edges from v to w in G.adjacentEdges(v) do

4 if vertex w is not labeled as discovered then

5 recursively call DFS(G,w)

A non-recursive implementation of DFS:

1 procedure DFS-iterative(G,v):

2 let S be a stack

3 S.push(v)

4 while S is not empty

5 v = S.pop()

6 if v is not labeled as discovered:

7 label v as discovered

8 for all edges from v to w in G.adjacentEdges(v) do

9 S.push(w)

## Iterative Deepening Depth-first search:

Iterative Deepening Depth-first search (ID-DFS) is a state space/graph search strategy in which a depth-limited version of depth-first search is run repeatedly with increasing depth limits until the goal is found. IDDFS is equivalent to breadth-first search, but uses much less memory; on each iteration, it visits the nodes in the search tree in the same order as depth-first search, but the cumulative order in which nodes are first visited is effectively breadth-first.

## Pseudocode:

Set all nodes to "not visited";

s = new Stack(); \*\*\*\*\*\*\* Change to use a stack

s.push(initial node); \*\*\*\*\* Push() stores a value in a stack

while ( s ≠ empty ) do

{

x = s.pop(); \*\*\*\*\*\* Pop() remove a value from the stack

if ( x has not been visited )

{

visited[x] = true; // Visit node x !

for ( every edge (x, y) /\* we are using all edges ! \*/ )

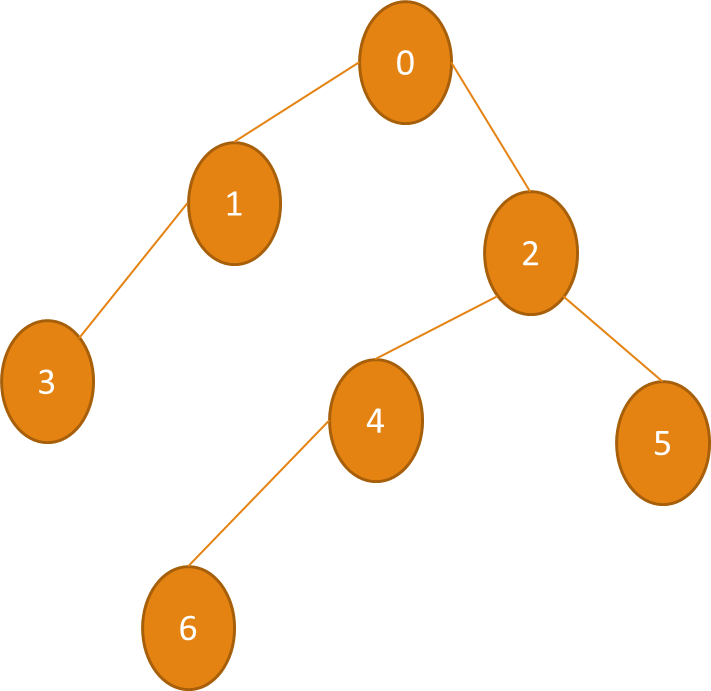
if ( y has not been visited )

s.push(y); \*\*\*\*\* Use push() !

}

}

### Example:



### Code:

vertexList = [**'0'**, **'1'**, **'2'**, **'3'**, **'4'**, **'5'**, **'6'**]  
edgeList = [(0,1), (0,2), (1,0) , (1,3) , (2,0) , (2,4) , (2,5) , (3,1), (4,2) , (4,6), (5,2), (6,4)]  
graphs = (vertexList, edgeList)  
  
def dfs(graph, start):  
 vertexList, edgeList = graph  
 visitedVertex = []  
 stack = [start]  
 adjacencyList = [[] for vertex in vertexList]  
  
 for edge in edgeList:  
 adjacencyList[edge[0]].append(edge[1])  
  
 while stack:  
 current = stack.pop()  
 for neighbor in adjacencyList[current]:  
 if not neighbor in visitedVertex:  
 stack.append(neighbor)  
 visitedVertex.append(current)  
 return visitedVertex  
  
print(dfs(graphs, 0))

# Breadth-first search:

Breadth-first search (BFS) is an algorithm that is used to graph data or searching tree or traversing structures. The full form of BFS is the Breadth-first search.

The algorithm efficiently visits and marks all the key nodes in a graph in an accurate breadthwise fashion. This algorithm selects a single node (initial or source point) in a graph and then visits all the nodes adjacent to the selected node. Remember, BFS accesses these nodes one by one.

Once the algorithm visits and marks the starting node, then it moves towards the nearest unvisited nodes and analyses them. Once visited, all nodes are marked. These iterations continue until all the nodes of the graph have been successfully visited and marked.

## Graph traversals

A graph traversal is a commonly used methodology for locating the vertex position in the graph. It is an advanced search algorithm that can analyze the graph with speed and precision along with marking the sequence of the visited vertices. This process enables you to quickly visit each node in a graph without being locked in an infinite loop.

## How BFS works:

1. Graph traversal requires the algorithm to visit, check, and/or update every single un-visited node in a tree-like structure. Graph traversals are categorized by the order in which they visit the nodes on the graph.
2. BFS algorithm starts the operation from the first or starting node in a graph and traverses it thoroughly. Once it successfully traverses the initial node, then the next non-traversed vertex in the graph is visited and marked.
3. Hence, you can say that all the nodes adjacent to the current vertex are visited and traversed in the first iteration. A simple queue methodology is utilized to implement the working of a BFS algorithm

## Pseudocode:

Set all nodes to "not visited";

q = new Queue();

q.enqueue(initial node);

while ( q ≠ empty ) do

{

x = q.dequeue();

if ( x has not been visited )

{

visited[x] = true; // Visit node x !

for ( every edge (x, y) /\* we are using all edges ! \*/ )

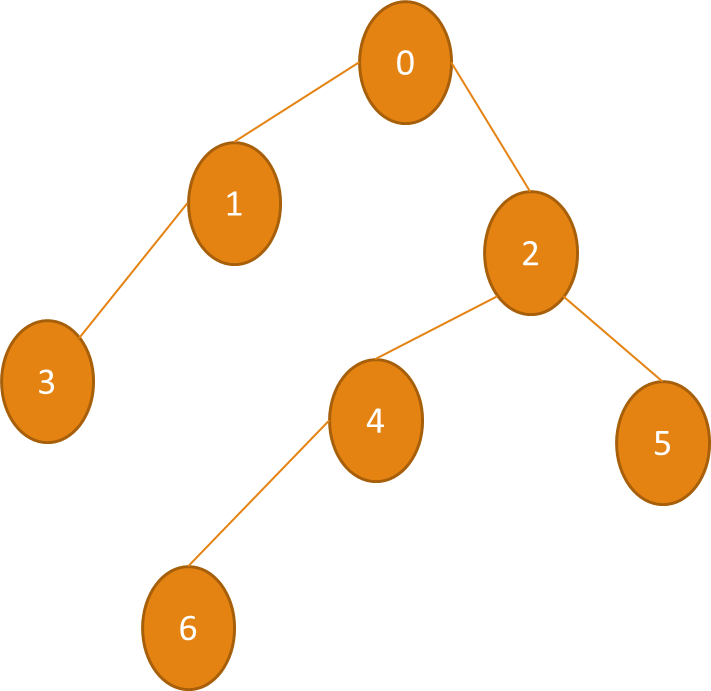
if ( y has not been visited )

q.enqueue(y); // Use the edge (x,y) !!!

}

}

### Example:

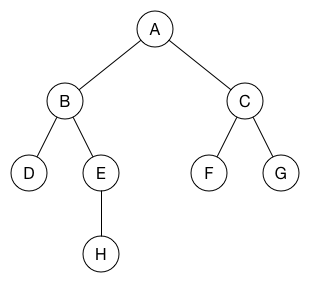


### Code:

vertexList = [**'0'**, **'1'**, **'2'**, **'3'**, **'4'**, **'5'**, **'6'**]  
edgeList = [(0,1), (0,2), (1,0) , (1,3) , (2,0) , (2,4) , (2,5) , (3,1), (4,2) , (4,6), (5,2), (6,4)]  
graphs = (vertexList, edgeList)  
def bfs(graph, start):  
 vertexList, edgeList = graph  
 visitedList = []  
 queue = [start]  
 adjacencyList = [[] for vertex in vertexList]  
  
 *# fill adjacencyList from graph* for edge in edgeList:  
 adjacencyList[edge[0]].append(edge[1])  
  
 *# bfs* while queue:  
 current = queue.pop()  
 for neighbor in adjacencyList[current]:  
 if not neighbor in visitedList:  
 queue.insert(0,neighbor)  
 visitedList.append(current)  
 return visitedList  
  
print(bfs(graphs, 0))

# TASKS

**TASK:** Consider the below graph;



Use BFS to find the shortest path between A and F. (Hint: the distance between any consecutive vertices is 1, i.e. distance between A and D is 2 ((A to B=1) + (B to D=1) = 2)

TASK: Consider the below graph;



Using DFS, check if there is any path exists between any two nodes? Also the return the path.

e.g. If user two vertices i.e. 2 and 1; the program should return : Yes the paths exist, which are [2,1],[2,0,1].

# **LAB 7: Introduction to NumPy, Pandas, Scikit-learn and Matplotlib Python Packages**

# Objectives:

1. To learn about Python most widely used libraries in machine learning

# Different Python Packages

# NUMPY

NumPy is the cornerstone toolbox for scientific computing with Python. NumPy provides, among other things, support for multidimensional arrays with basic operations on them and useful linear algebra functions. Many toolboxes use the NumPy array representations as an efficient basic data structure.

### Examples

#importing numpy package

mport numpy as np  
b=np.array([[[1,2,3,5],[2,3,4,4]],[[1,2,3,5],[2,3,4,4]]])  
  
#printing the data type  
print(b.dtype)

out: int32

#printing the dimension of the NumPy array  
print(b.ndim)

Out: 3

#printing the shape the NumPy array  
print(b.shape)

Out: (2, 2, 4)

# printing the size the NumPy array i.e. total number of elements  
print(b.size)  
out: 16

#to generate an array of numerical numbers from 10 to 100 with 2 steps  
c=np.arange(10,100,2)  
print(c)

#to generate an array of zeros  
b=np.zeros(10)

print(b)

#to generate a array of ones  
b=np.ones(10)  
print(b)

#to shuffle data

a=np.random.permutation(c)  
print(a)

#to generate random uniform noise  
d=np.random.rand(1000)

#to generate random gaussian noise  
d=np.random.randn(1000)

#to select random integer

f=np.random.randint(10,40)

#to reshape an NumPy array  
d=np.arange(20).reshape(4,5)  
print(d)

#slicing  
print(d[1:3,2:4])

# SCIKIT-Learn

Scikit-learn is a machine learning library built from NumPy, SciPy, and Matplotlib. Scikit-learn offers simple and efficient tools for common tasks in data analysis such as classification, regression, clustering, dimensionality reduction, model selection, and preprocessing.

### Example

#Loading an example dataset

from sklearn import datasets

iris = datasets.load\_iris()

digits = datasets.load\_digits()

# PANDAS

Pandas5 provides high-performance data structures and data analysis tools. The key feature of Pandas is a fast and efficient DataFrame object for data manipulation with integrated indexing. The DataFrame structure can be seen as a spreadsheet which offers very flexible ways of working with it. You can easily transform any dataset in the way you want, by reshaping it and adding or removing columns or rows. It also provides high-performance functions for aggregating, merging, and joining datasets. Pandas also has tools for importing and exporting data from different formats: comma-separated value (CSV), text files, Microsoft Excel, SQL databases, and the fast HDF5 format. In many situations, the data you have in such formats will not be complete or totally structured. For such cases, Pandas offers handling of missing data and intelligent data alignment. Furthermore, Pandas provides a convenient Matplotlib interface.

### Examples

#importing pandas library

import pandas as pd

#creating Pandas series  
a=pd.Series([1,2,3,4],index=[**'a'**,**'b'**,**'c'**,**'d'**])  
print(a)

marks={**"A"**:10,**"B"**:20,**"C"**:30}  
print(marks)  
grades={**"A"**:2,**"B"**:3,**"C"**:5}

#converting dictionaries to the Pandas series

pd1=pd.Series(marks)  
print(pd1)  
pd2=pd.Series(grades)  
*#print(marks)  
#print(pd1)*

*#Pandas DataFrame*pd3=pd.DataFrame({**"marks"**:pd1,**"grades"**:pd2})  
print(pd3)

#adding dictionary to the Pandas DataFrame  
pd3[**"percentage"**]=pd3[**"marks"**]/100  
print(pd3)

#deleting from Pandas Dataframe  
del pd3[**"percentage"**]

#Thresholding  
print(pd3[pd3[**'marks'**]>10])  
print(pd3)

### 

# Matplotlib

Matplotlib produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shell, web application servers, and various graphical user interface toolkits. matplotlib.pyplot is a collection of functions that make matplotlib work like MATLAB. Majority of plotting commands in pyplot have MATLAB analogs with similar arguments.

### Example

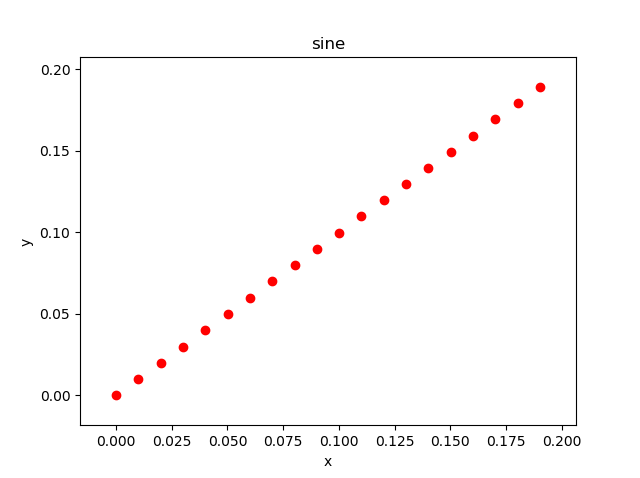
#importing Matplotlib.Pyplot

import matplotlib.pyplot as plt  
x=np.linspace(0,10,1000)

#conitonus plotting

plt.plot(x,np.sin(x), color=**"red"**)

# Discrete Plotting  
plt.scatter(x[0:20],np.sin(x[0:20]), color=**"red"**)  
plt.xlabel(**"x"**)  
plt.ylabel(**"y"**)  
plt.title(**"sine"**)  
plt.show()



# TASKS

TASK 1:

Write a NumPy program to create a random 10x4 array and extract the first five rows of the array and store them into a variable.

TASK 2:

Write a Pandas program to select the rows where the number of attempts in the examination is greater than 2.   
Sample Python dictionary data and list labels:  
exam\_data = {'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas'],  
'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],  
'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],  
'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']}  
labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']  
Expected Output:  
Number of attempts in the examination is greater than 2:  
name score attempts qualify  
b Dima 9.0 3 no  
d James NaN 3 no  
f Michael 20.0 3 yes

TASK 3:

From the sample data given in TASK 2; write a program to calculate the average of the scores. The program should be able to ignore NaN values.

Expected Output:

The average score is : 13.56

# **LAB 8: Introduction to Machine Learning, Deep learning and deep learning Frameworks (TensorFlow, Keras) in Python**

# Objectives:

1. To understand machine learning and deep learning and deep learning frameworks in Python

# Machine Learning and deep learning

Our imaginations have long been captivated by visions of machines that can learn and imitate human intelligence. Software programs that can acquire new knowledge and skills through experience are becoming increasingly common. We use such machine learning programs to discover new music that we might enjoy, and to find exactly the shoes we want to purchase online. Machine learning programs allow us to dictate commands to our smart phones, and allow our thermostats to set their own temperatures. Machine learning programs can decipher sloppily-written mailing addresses better than humans, and can guard credit cards from fraud more vigilantly. From investigating new medicines to estimating the page views for versions of a headline, machine learning software is becoming central to many industries. Machine learning has even encroached on activities that have long been considered uniquely human, such as writing the sports column recapping the Duke basketball team's loss to UNC.

# Learning from experience

Machine learning systems are often described as learning from experience either with or without supervision from humans. In supervised learning problems, a program predicts an output for an input by learning from pairs of labeled inputs and outputs. That is, the program learns from examples of the "right answers". In unsupervised learning, a program does not learn from labeled data. Instead, it attempts to discover patterns in data. For example, assume that you have collected data describing the heights and weights of people. An example of an unsupervised learning problem is dividing the data points into groups. A program might produce groups that correspond to men and women, or children and adults. Now assume that the data is also labeled with the person's sex. An example of a supervised learning problem is to induce a rule for predicting whether a person is male or female based on his or her height and weight. We will discuss algorithms and examples of supervised and unsupervised learning in the following chapters.

# Machine learning tasks

Two of the most common **supervised machine learning** tasks are **classification** and **regression**. In classification tasks, the program must learn to predict discrete values for one or more response variables from one or more features. That is, the program must predict the most probable category, class, or label for new observations.

**Applications** of classification include predicting whether a stock's price will rise or fall, or deciding whether a news article belongs to the politics or leisure sections. In regression problems, the program must predict the values of one more or continuous response variables from one or more features.

**Examples** of regression problems include predicting the sales revenue for a new product, or predicting the salary for a job based on its description. Like classification, regression problems require supervised learning.

A common **unsupervised learning task** is to discover groups of related observations, called **clusters**, within the dataset. This task, called clustering or cluster analysis, assigns observations into groups such that observations within a group are more similar to each other based on some similarity measure than they are to observations in other groups.

Clustering is often used to explore a dataset. For example, given a collection of movie reviews, a clustering algorithm might discover the sets of positive and negative reviews. The system will not be able to label the clusters as positive or negative; without supervision, it will only have knowledge that the grouped observations are similar to each other by some measure. A common application of clustering is discovering segments of customers within a market for a product. By understanding what attributes are common to particular groups of customers, marketers can decide what aspects of their campaigns to emphasize. Clustering is also used by internet radio services; given a collection of songs, a

clustering algorithm might be able to group the songs according to their genres. Using different similarity measures, the same clustering algorithm might group the songs by their keys, or by the instruments they contain.

# Training data, testing data, and validation data

A **training set** is a collection of observations. These observations comprise the experience that the algorithm uses to learn. In supervised learning problems, each observation consists of an observed response variable and features of one or more observed explanatory variables. The **test set** is a similar collection of observations. The test set is used to evaluate the performance of the model using some performance metric. It is important that no observations from the training set are included in the test set. If the test set does contain examples from the training set, it will be difficult to assess whether the

algorithm has learned to generalize from the training set or has simply memorized it. A program that generalizes well will be able to effectively perform a task with new data.

In addition to the training and test data, a third set of observations, called a **validation or hold-out set**, is sometimes required. The validation set is used to tune variables called hyperparameters that control how the algorithm learns from the training data. The program is still evaluated on the test set to provide an estimate of its performance in the real world. The validation set should not be used to estimate real-world performance because the program has been tuned to learn from the training data in a way that optimizes its score on the validation data; the program will not have this advantage in the real world.

# Deep Learning

Deep learning is a specific subset of Machine Learning, which is a specific subset of Artificial Intelligence. For individual definitions:

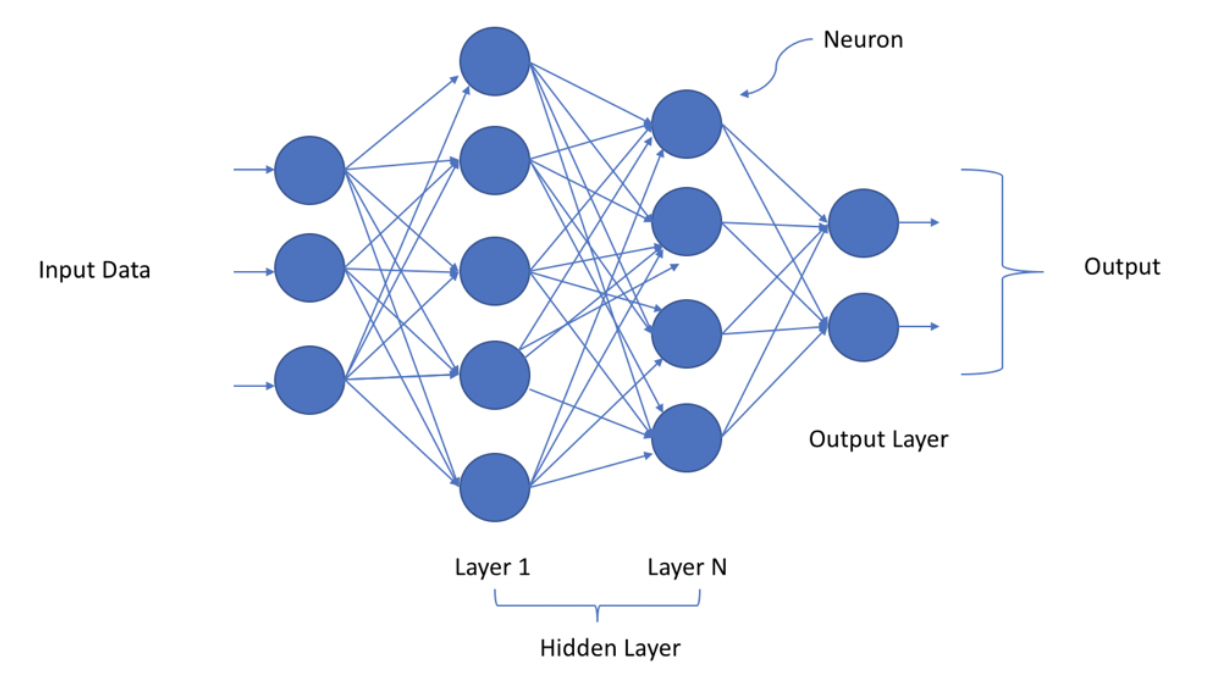
* Artificial Intelligence is the broad mandate of creating machines that can think intelligently
* Machine Learning *is one way of doing that*, by using algorithms to glean insights from data (see our gentle introduction here)
* Deep Learning *is one way of doing that*, using a specific algorithm called a Neural Network

Don’t get lost in the taxonomy – Deep Learning is just a type of algorithm that seems to work really well for predicting things. Deep Learning and Neural Nets, for most purposes, are effectively synonymous. If people try to confuse you and argue about technical definitions, don’t worry about it: like Neural Nets, labels can have many layers of meaning.

Neural networks are [inspired by the structure of the cerebral cortex](http://pages.cs.wisc.edu/~bolo/shipyard/neural/local.html). At the basic level is the perceptron, the mathematical representation of a biological neuron. Like in the cerebral cortex, there can be several layers of interconnected perceptrons.

Input values, or in other words our underlying data, get passed through this “network” of hidden layers until they eventually converge to the output layer. The output layer is our prediction: it might be one node if the model just outputs a number, or a few nodes if it’s a multiclass classification problem.

The hidden layers of a Neural Net perform modifications on the data to eventually feel out what its relationship with the target variable is. Each node has a weight, and it multiplies its input value by that weight. Do that over a few different layers, and the Net is able to essentially manipulate the data into something meaningful. To figure out what these small weights should be, we typically use [an algorithm called Backpropagation](https://brilliant.org/wiki/backpropagation/). For more details visit: <https://algorithmia.com/blog/introduction-to-deep-learning>



# What is TensorFlow?

“TensorFlow is an open-source machine learning library for research and production. TensorFlow offers APIs for beginners and experts to develop for desktop, mobile, web, and cloud.” - [TensorFlow Website](https://www.tensorflow.org/tutorials/)

# What is Keras?

“Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.” - [keras.io](https://keras.io/)

# How to create deep learning environment?

Visit these links for better deep learning environment creation;

1. <https://towardsdatascience.com/setup-an-environment-for-machine-learning-and-deep-learning-with-anaconda-in-windows-5d7134a3db10>
2. <https://towardsdatascience.com/installing-keras-tensorflow-using-anaconda-for-machine-learning-44ab28ff39cb>

<https://www.pyimagesearch.com/2016/11/14/installing-keras-with-tensorflow-backend/>

# TASK:

1. Difference Supervised machine learning and Unsupervised machine learning?
2. Difference between Classification problem and Regression problem?
3. Difference between machine learning and deep learning?
4. Difference between Keras and Tensorflow frameworks?

# **LAB 9: Supervised machine Learning: Classification with K-Nearest Neighbors**

# Objectives:

1. To learn and Implement K-Nearest Neighbor machine learning technique

# K-Nearest Neighbors

KNN is a simple model for regression and classification tasks. It is so simple that its name describes most of its learning algorithm. The titular neighbors are representations of training instances in a metric space. A metric space is a feature space in which the distances between all members of a set are defined. In the previous chapter's pizza problem, our training instances were represented in a metric space because the distances between all the pizza diameters was defined. These neighbors are used to estimate the value of the response variable for a test instance. The hyperparameter k specifies how many neighbors can be used in the estimation. A hyperparameter is a parameter that controls how the algorithm learns; hyperparameters are not estimated from the training data and are sometimes set manually. Finally, the k neighbors that are selected are those that are nearest to the test instance, as measured by some distance function.

For classification tasks, a set of tuples of feature vectors and class labels comprise the training set. KNN is a capable of binary, multi-class, and multi-label classification; we will define these tasks later, and we will focus on binary classification in this chapter. The simplest KNN classifiers use the mode of the KNN labels to classify test instances, but other strategies can be used. The k is often set to an odd number to prevent ties.

In regression tasks, the feature vectors are each associated with a response variable that takes a real-valued scalar instead of a label. The prediction is the mean or weighted mean of the KNN response variables.

### Example:

Suppose there are some shaded red and blue circles that are labeled as ‘+’ and ‘-’ respectively, as shown in Figure 4. In order to determine the class of a test sample shown as ‘?’ with k=5, the votes of five nearest neighbors are considered. In this case, as nearest neighbors to both test samples are mostly red shaded circles, so both of them are identified as ‘+’.

Figure 1: K-NN example

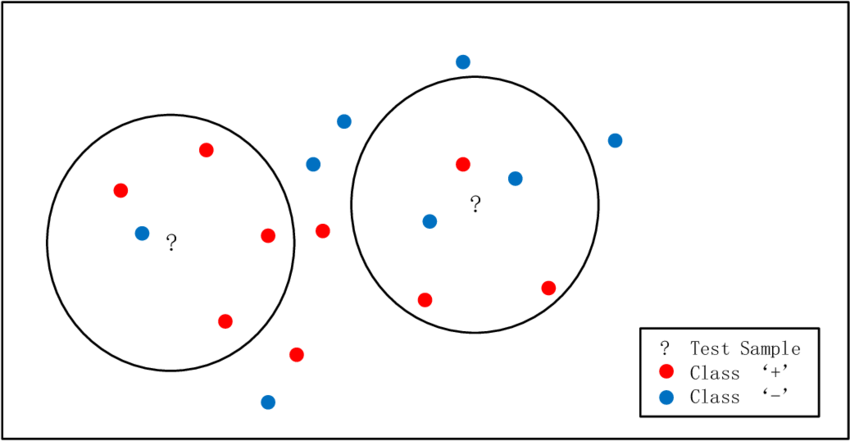


Fig. 4: K-nearest neighbour

# Implementation of K-NN Classifier

### Dataset

The data set contains 3 classes of 50 instances each, where each class 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. class:  
-- Iris Setosa  
-- Iris Versicolour  
-- Iris Virginica

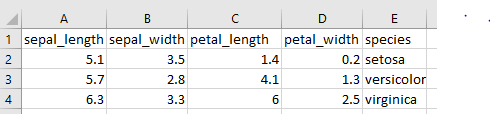


Figure 2: sample of dataset

import numpy as np  
import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import StandardScaler  
data=pd.read\_csv(**r'F:\path\iris\_data\_2.csv'**)  
print(data.head())  
  
x=data.drop(**'species'**,**'columns'**)  
y=data[**'species'**]  
  
x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.20)  
*#print(x\_train)*scaler=StandardScaler()  
scaler.fit(x\_train)  
*#print(x\_train)*x\_train=scaler.transform(x\_train)  
x\_test=scaler.transform(x\_test)  
print(x\_train)  
print(x\_test)  
from sklearn.neighbors import KNeighborsClassifier   
classifier=KNeighborsClassifier(n\_neighbors=3)  
classifier.fit(x\_train,y\_train)  
result=classifier.predict(x\_test)  
print(result)  
from sklearn.metrics import confusion\_matrix,classification\_report  
print(classification\_report(y\_test,result))  
print(confusion\_matrix(y\_test,result))

# TASK:

Implement support vector machine for the same classification problem

1. Attach the screenshot of the code and output

# **LAB 10: Supervised machine Learning: Regression with K-Nearest Neighbors**

# Objectives:

* + 1. To learn and Implement K-Nearest Neighbor regressor

# K-Nearest Neighbors

KNN is a simple model for regression and classification tasks. It is so simple that its name describes most of its learning algorithm. The titular neighbors are representations of training instances in a metric space. A metric space is a feature space in which the distances between all members of a set are defined. In the previous chapter's pizza problem, our training instances were represented in a metric space because the distances between all the pizza diameters was defined. These neighbors are used to estimate the value of the response variable for a test instance. The hyperparameter k specifies how many neighbors can be used in the estimation. A hyperparameter is a parameter that controls how the algorithm learns; hyperparameters are not estimated from the training data and are sometimes set manually. Finally, the k neighbors that are selected are those that are nearest to the test instance, as measured by some distance function.

For classification tasks, a set of tuples of feature vectors and class labels comprise the training set. KNN is a capable of binary, multi-class, and multi-label classification; we will define these tasks later, and we will focus on binary classification in this chapter. The simplest KNN classifiers use the mode of the KNN labels to classify test instances, but other strategies can be used. The k is often set to an odd number to prevent ties.

In regression tasks, the feature vectors are each associated with a response variable that takes a real-valued scalar instead of a label. The prediction is the mean or weighted mean of the KNN response variables.

### Example:

Suppose there are some shaded red and blue circles that are labeled as ‘+’ and ‘-’ respectively, as shown in Figure 4. In order to determine the class of a test sample shown as ‘?’ with k=5, the votes of five nearest neighbors are considered. In this case, as nearest neighbors to both test samples are mostly red shaded circles, so both of them are identified as ‘+’.

Figure 1: K-NN example

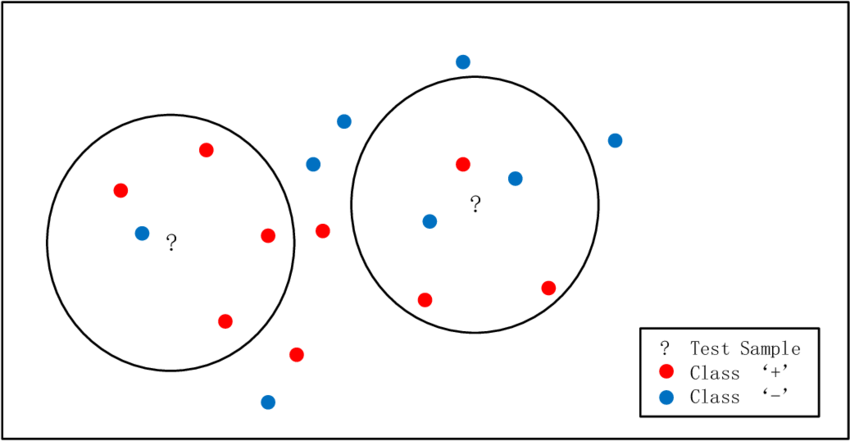


Fig. 4: K-nearest neighbour

# Implementation of K-NN Regressor

### Dataset

The data set contains 3 classes of 50 instances each, where each class 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. class:  
-- 1.2 (Iris Setosa)   
-- 2.3 (Iris Versicolour)  
-- 3.5 (Iris Virginica)

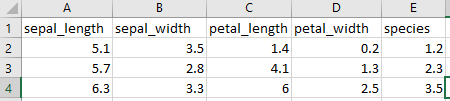


Figure 2: sample of dataset

import numpy as np  
import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import StandardScaler  
data=pd.read\_csv(**r'F:\path\iris\_data\_2.csv'**)  
print(data.head())  
  
x=data.drop(**'species'**,**'columns'**)  
y=data[**'species'**]  
  
x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.20)  
*#print(x\_train)*scaler=StandardScaler()  
scaler.fit(x\_train)  
*#print(x\_train)*x\_train=scaler.transform(x\_train)  
x\_test=scaler.transform(x\_test)  
print(x\_train)  
print(x\_test)  
from sklearn.neighbors import KNeighborsRegressor  
regressor=KNeighborsRegressor(n\_neighbors=3)  
regressor.fit(x\_train,y\_train)  
result= regressor.predict(x\_test)  
print(result)  
from sklearn.metrics import confusion\_matrix,classification\_report  
print(classification\_report(y\_test,result))  
print(confusion\_matrix(y\_test,result))

# TASK:

Implement linear regression for the same regression problem

1. Attach the screenshot of the code and output

# **LAB 11: Unsupervised machine learning: K-mean clustering**

# Objectives:

1. To learn and Implement K-mean clustering machine learning technique

# Unsupervised machine learning

In machine learning, the problem of unsupervised learning is that of trying to find hidden structure in unlabeled data. Since the examples given to the learner are unlabeled, there is no error or reward signal to evaluate the goodness of a potential solution. This distinguishes unsupervised from supervised learning. Unsupervised learning is defined as the task performed by algorithms that learn from a training set of unlabeled or unannotated examples, using the features of the inputs to categorize

them according to some geometric or statistical criteria. Unsupervised learning encompasses many techniques that seek to summarize and explain key features or structures of the data. Many methods employed in unsupervised learning are based on data mining methods used to preprocess data. Most unsupervised learning techniques can be summarized as those that tackle the following four groups of problems:

• **Clustering:** has as a goal to partition the set of examples into groups.

• **Dimensionality reduction:** aims to reduce the dimensionality of the data. Here, we encounter techniques such as Principal Component Analysis (PCA), independent component analysis, and nonnegative matrix factorization.

• **Outlier detection:** has as a purpose to find unusual events (e.g., a malfunction), that distinguish part of the data from the rest according to certain criteria.

• **Novelty detection:** deals with cases when changes occur in the data (e.g., in streaming data). The most common unsupervised task is clustering, which we focus on in this Lab.

# Clustering

Clustering is a process of grouping similar objects together; i.e., to partition unlabeled examples into disjoint subsets of clusters, such that:

• Examples within a cluster are similar (in this case, we speak of high intraclass similarity).

• Examples in different clusters are different (in this case, we speak of low interclass similarity). When we denote data as similar and dissimilar, we should define a measure for this similarity/dissimilarity. Note that grouping similar data together can help in discovering new categories in an unsupervised manner, even when no sample category labels are provided. Moreover, two kinds of inputs can be used for grouping:

(a) in similarity-based clustering, the input to the algorithm is an n × n dissimilarity matrix or distance matrix;

(b) in feature-based clustering, the input to the algorithm is an n × D feature matrix or design matrix, where n is the number of examples in the dataset and D the dimensionality of each sample.

# K-means

The K-means algorithm is a clustering method that is popular because of its speed and scalability. K-means is an iterative process of moving the centers of the clusters, called the **centroids**, to the mean position of their constituent instances and re-assigning instances to the clusters with the closest centroids. The titular *k* is a hyperparameter that specifies the number of clusters that should be created; K-means automatically assigns observations to clusters but cannot determine the appropriate number of clusters. *k* must be a positive integer that is less than the number of instances in the training set. Sometimes the number of clusters is specified by the clustering problem's context. For example, a company that manufactures shoes might know that it is able to support manufacturing three new models.

To understand what groups of customers to target with each model, it surveys customers and creates three clusters from the results, that is, the number of clusters specified by the problem's context. Other problems may not require a specific number of clusters, and the optimal number of clusters may be ambiguous.

The parameters of K-means are the positions of the clusters' centroids and the observations

that are assigned to each cluster. Like generalized linear models and decision trees, the

optimal values of K-means' parameters are found by minimizing a cost function. The cost

function for K-means is given by the following equation:



Here, *μk* is the centroid for cluster *k*. The cost function sums the distortions of the clusters.

Each cluster's distortion is equal to the sum of the squared distances between its centroid

and its constituent instances. The distortion is small for compact clusters and large for

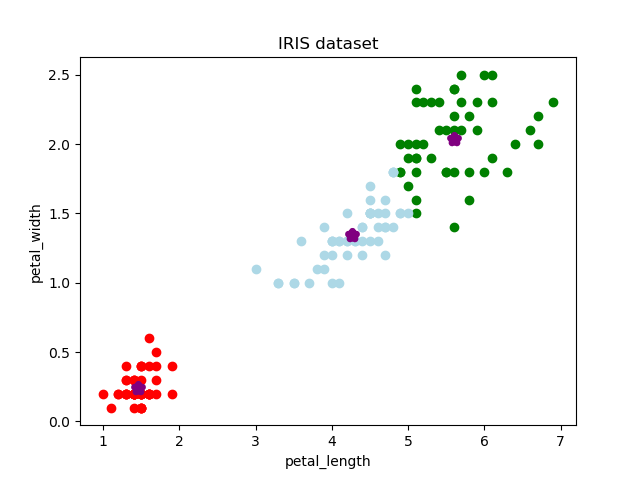
clusters that contain scattered instances. The parameters that minimize the cost function are

learned through an iterative process of assigning observations to clusters and then moving

the clusters. First, the clusters' centroids are initialized, often by randomly selecting

instances. During each iteration, K-means assigns observations to the cluster that they are

closest to and then moves the centroids to their assigned observations' mean location.



# TASK:

Use sepal\_length and sepal\_width as features from the iris\_dataset and apply K-mean clustering. Attach the code and graph.

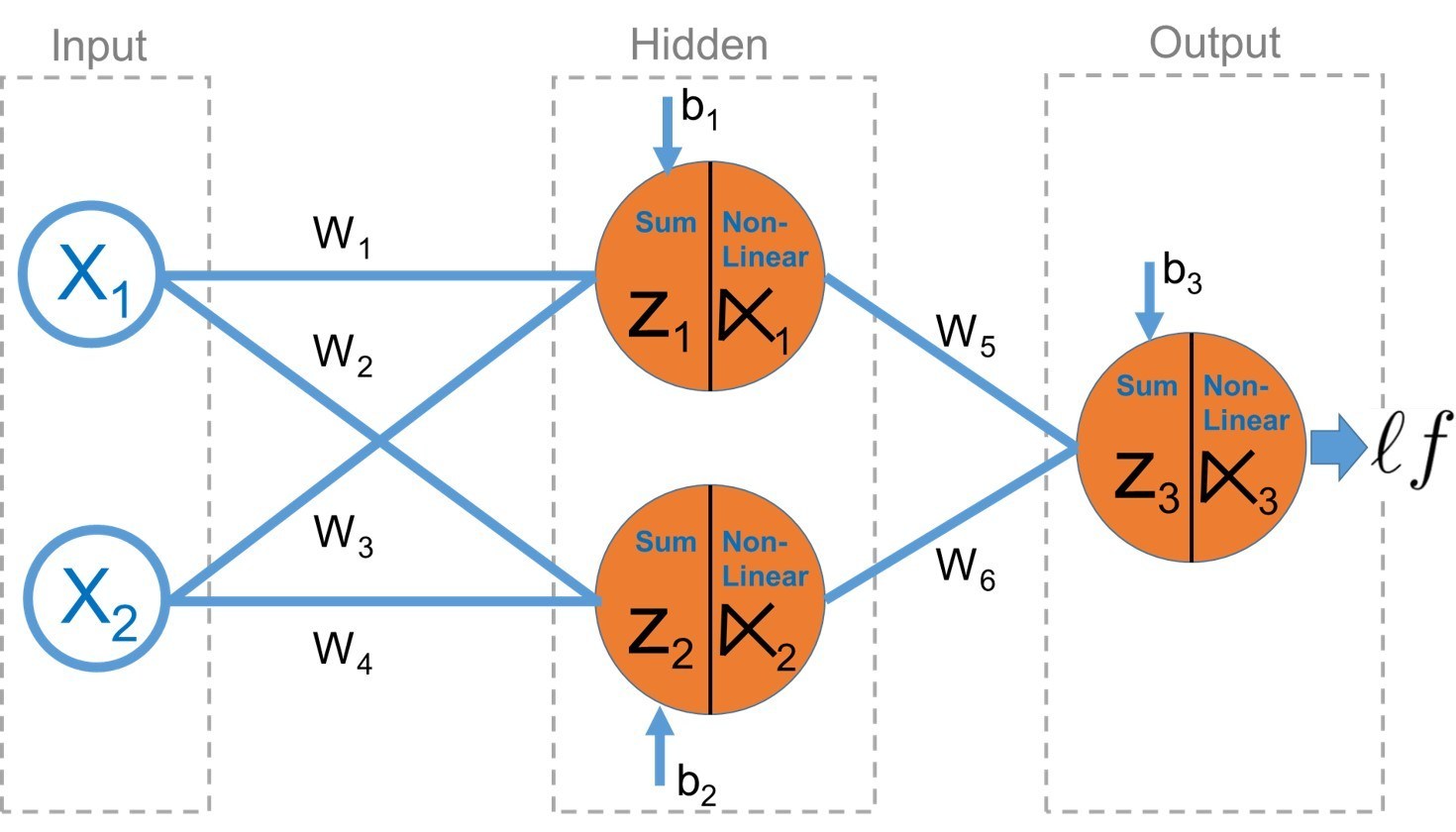
# **LAB 12: Implementation of Neural Networks (NN) in Python**

# Objectives:

1. To learn and implement ANN in Python

# Artificial Neural Networks

An Artificial Neural Network (ANN) is a computational model and architecture that simulates biological neurons and the way they function in our brain. Typically, an ANN has layers of interconnected nodes. The nodes and their inter-connections are analogous to the network of neurons in our brain. A typical ANN has an input layer, an output layer, and at least one hidden layer between the input and output with inter-connections, as depicted in Figure below.



Any basic ANN will always have multiple layers of nodes, specific connection patterns and links between the layers, connection weights and activation functions for the nodes/neurons that convert weighted inputs to outputs. The process of learning for the network typically involves a cost function and the objective is to optimize the cost function (typically minimize the cost). The weights keep getting updated in the process of learning.

# Backpropagation

The backpropagation algorithm is a popular technique to train ANNs and it led to a resurgence in the popularity of neural networks in the 1980s. The algorithm typically has two main stages—propagation and weight updates. They are described briefly as follows.

1. Propagation

a. The input data sample vectors are propagated forward through the neural network to generate the output values from the output layer.

b. Compare the generated output vector with the actual/desired output vector for that input data vector.

c. Compute difference in error at the output units.

d. Backpropagate error values to generate deltas at each node/neuron.

2. Weight Update

a. Compute weight gradients by multiplying the output delta (error) and input activation.

b. Use learning rate to determine percentage of the gradient to be subtracted from original weight and update the weight of the nodes.

These two stages are repeated multiple times with multiple iterations/epochs until we get satisfactory results. Typically, backpropagation is used along with optimization algorithms or functions like stochastic gradient descent.

# Multilayer Perceptrons

A multilayer perceptron, also known as MLP, is a fully connected, feed-forward artificial neural network with at least three layers (input, output, and at least one hidden layer) where each layer is fully connected to the adjacent layer. Each neuron usually is a non-linear functional processing unit. Backpropagation is typically used to train MLPs and even deep neural nets are MLPs when they have multiple hidden layers. Typically used for supervised Machine Learning tasks like classification.

# Activation function

In artificial neural networks, the activation function of a node defines the output of that node given an input or set of inputs. A standard integrated circuit can be seen as a digital network of activation functions that can be "ON" or "OFF", depending on input.

# Loss function

In mathematical optimization and decision theory, a loss function or cost function is a function that maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event. An optimization problem seeks to minimize a loss function.

# Optimizers

Optimizers are algorithms or methods used to change the attributes of the neural network such as weights and learning rate to reduce the losses. Optimizers are used to solve optimization problems by minimizing the function.

# Implementation:

import pandas as pd  
data=pd.read\_csv(**r'F:\path\iris\_data\_2.csv'**)  
data1=data.drop(**'species'**,**'columns'**)  
labels=data[**'species'**]  
  
from sklearn.model\_selection import train\_test\_split  
train\_x,test\_x,train\_y,test\_y=train\_test\_split(data1,labels, test\_size=0.30)  
  
import keras  
from keras.models import Sequential  
from keras.layers import Dense  
model=Sequential()  
model.add(Dense(32,input\_dim=4,activation=**'relu'**))  
model.add(Dense(1,activation=**'sigmoid'**))  
print(model.summary())  
  
model.compile(optimizer=**'adam'**, loss=**'binary\_crossentropy'**, metrics=[**'accuracy'**])  
history=model.fit(train\_x,train\_y,epochs=5, batch\_size=1, validation\_split=0.10)  
results=model.predict\_classes(test\_x)  
print(results)  
print(history.history.keys())  
import matplotlib.pyplot as plt  
plt.plot(history.history[**'accuracy'**])  
plt.plot(history.history[**'val\_accuracy'**])  
plt.title(**'model accuracy'**)  
plt.xlabel(**'epochs'**)  
plt.ylabel(**'accuracy'**)  
plt.legend([**'train'**,**'valid'**])  
plt.show()  
plt.plot(history.history[**'loss'**])  
plt.plot(history.history[**'val\_loss'**])  
plt.title(**'model loss'**)  
plt.xlabel(**'epochs'**)  
plt.ylabel(**'loss'**)  
plt.legend([**'train'**,**'valid'**])  
plt.show()  
from sklearn.metrics import accuracy\_score  
print(**'accuracy'**,accuracy\_score(test\_y,results))  
from sklearn.metrics import confusion\_matrix,classification\_report  
print(classification\_report(test\_y,results))  
print(confusion\_matrix(test\_y,results))

# TASK:

**Change the number of the layers and neurons and observe if the score improves or not. Share the graphs.**

# **LAB 13: Evaluation Metrics to evaluate machine learning algorithms**

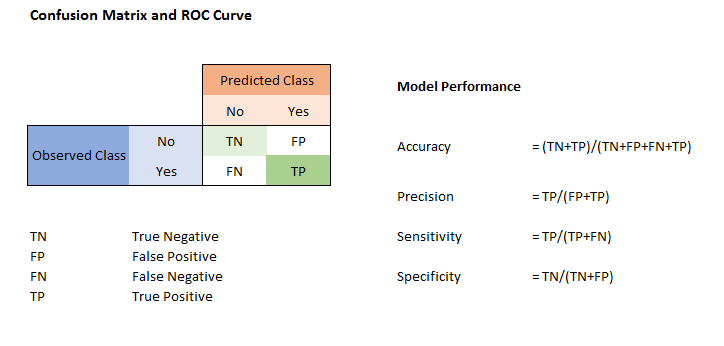
# Objectives:

1. To learn evaluation metrics to evaluate machine learning algorithms

# Confusion Matrix

Confusion Matrix as the name suggests gives us a matrix as output and describes the complete performance of the model.

Lets assume we have a binary classification problem. We have some samples belonging to two classes : YES or NO.



Confusion Matrix

There are 4 important terms :

* **True Positives** : The cases in which we predicted YES and the actual output was also YES.
* **True Negatives** : The cases in which we predicted NO and the actual output was NO.
* **False Positives** : The cases in which we predicted YES and the actual output was NO.
* **False Negatives** : The cases in which we predicted NO and the actual output was YES.

# Accuracy

Accuracy for the matrix can be calculated by taking average of the values lying across the**“main diagonal”**i.e

Image for post

Confusion Matrix forms the basis for the other types of metrics.

# Area Under Curve

Area Under Curve(AUC) is one of the most widely used metrics for evaluation. It is used for binary classification problem. AUC of a classifier is equal to the probability that the classifier will rank a randomly chosen positive example higher than a randomly chosen negative example. Before defining AUC, let us understand two basic terms :

* **True Positive Rate (Sensitivity)** : True Positive Rate is defined as TP/ (FN+TP). True Positive Rate corresponds to the proportion of positive data points that are correctly considered as positive, with respect to all positive data points.

Image for post

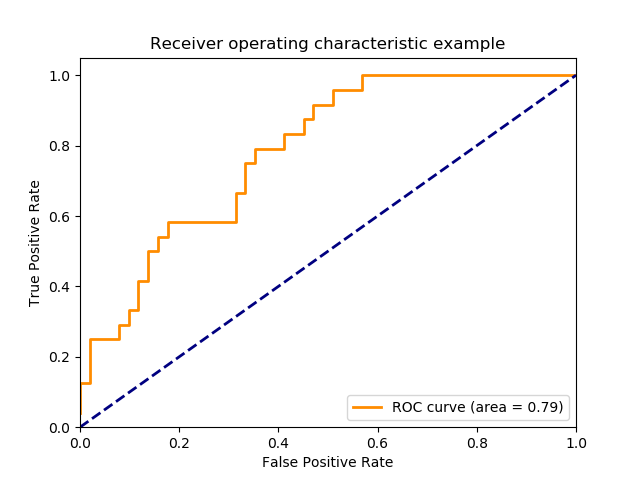
* **True Negative Rate (Specificity)** : True Negative Rate is defined as TN / (FP+TN). False Positive Rate corresponds to the proportion of negative data points that are correctly considered as negative, with respect to all negative data points.

Image for post

* **False Positive Rate**: False Positive Rate is defined as FP / (FP+TN). False Positive Rate corresponds to the proportion of negative data points that are mistakenly considered as positive, with respect to all negative data points.

Image for post

False Positive Rate and True Positive Rate both have values in the range **[0, 1]**. FPR and TPR both are computed at varying threshold values such as (0.00, 0.02, 0.04, …., 1.00) and a graph is drawn. AUC is the area under the curve of plot False Positive Rate vs True Positive Rate at different points in **[0, 1]**.



As evident, AUC has a range of [0, 1]. The greater the value, the better is the performance of our model.

# F1 Score

*F1 Score is used to measure a test’s accuracy*

F1 Score is the Harmonic Mean between precision and recall. The range for F1 Score is [0, 1]. It tells you how precise your classifier is (how many instances it classifies correctly), as well as how robust it is (it does not miss a significant number of instances).

High precision but lower recall, gives you an extremely accurate, but it then misses a large number of instances that are difficult to classify. The greater the F1 Score, the better is the performance of our model. Mathematically, it can be expressed as :

Image for post

F1 Score tries to find the balance between precision and recall.

* **Precision :**It is the number of correct positive results divided by the number of positive results predicted by the classifier.

Image for post

* **Recall :**It is the number of correct positive results divided by the number of **all**relevant samples (all samples that should have been identified as positive).

RecallImage for post

# Mean Absolute Error

Mean Absolute Error is the average of the difference between the Original Values and the Predicted Values. It gives us the measure of how far the predictions were from the actual output. However, they don’t gives us any idea of the direction of the error i.e. whether we are under predicting the data or over predicting the data. Mathematically, it is represented as :

Image for post

# Mean Squared Error

Mean Squared Error(MSE) is quite similar to Mean Absolute Error, the only difference being that MSE takes the average of the **square**of the difference between the original values and the predicted values. The advantage of MSE being that it is easier to compute the gradient, whereas Mean Absolute Error requires complicated linear programming tools to compute the gradient. As, we take square of the error, the effect of larger errors become more pronounced then smaller error, hence the model can now focus more on the larger errors.

Image for post

Source: <https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234>

# Python code

from sklearn.metrics import accuracy\_score, confusion\_matrix, roc\_auc\_score,plot\_roc\_curve, classification\_report  
cm=confusion\_matrix(y\_test,result)  
tn=cm[0,0]  
fp=cm[0,1]  
print(confusion\_matrix(y\_test,result))  
print(**'auc: '**,roc\_auc\_score(y\_test,result))  
plot\_roc\_curve(classifier,x\_test,y\_test)  
plt.show()  
print(classification\_report(y\_test,result))  
print(accuracy\_score(y\_test,result))