National University of Computer & Emerging Sciences, Karachi
Computer Science Department
Spring 2023, Lab Manual – 02

Course Code: AI-2002	Course : Artificial Intelligence Lab
Instructor(s):	Kariz Kamal, Saeeda Kanwal, Sania Urooj, Shakir Hussain, Omer Qureshi, Ali Fatimi

Contents:

Objective

- Introduction to Some Useful Python Libraries, understanding need of those libraries.
- To acquire programming skills in Python and its relevant libraries.
- Solve some basic AI problem using the python programming language.

Introduction to Python Libraries

A Python library is a reusable chunk of code that you may want to include in your programs/ projects. Compared to languages like C++ or C, a Python libraries do not pertain to any specific context in Python. Here, a 'library' loosely describes a collection of core modules. Essentially, then, a library is a collection of modules. A package is a library that can be installed using a package manager like rubygems or npm.

Python libraries list that will take you places in your journey with Artificial Intelligence. These are also the libraries for Data Science, Machine Learning, Deep Learning, Graph learningetc.

1. numpy

NumPy is a module for Python. The name is an acronym for "Numeric Python" or "NumericalPython". It is an extension module for Python, mostly written in C. This makes sure that theprecompiled mathematical and numerical functions and functionalities of Numpy guarantee great execution speed.
Furthermore, NumPy enriches the programming language Python with powerful data

structures, implementing multi-dimensional arrays and matrices. These data structures guarantee efficient calculations with matrices and arrays. The implementation is even aiming at huge matrices and arrays, better known under the heading of "big data". Besides that the module supplies a large library of high-level mathematical functions to operate on these matrices and arrays.

Code#1:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
print(type(arr))
#Create a 0-D array with value 42
arr = np.array(42)
print(arr)
#Create a 1-D array containing the values 1,2,3,4,5:
arr = np.array([1, 2, 3, 4, 5])
print(arr)
#Create a 2-D array containing two arrays with the values 1,2,3 and 4,5,6:
arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr)
#Create a 3-D array with two 2-D arrays, both containing two arrays with the values 1,2,3 and 4,5,6:
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(arr)
```

Code#2:

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[1])
print(arr[2] + arr[3])
#Access 2D array:
#Access the element on the first row, second column:
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('2nd element on 1st row: ', arr[0, 1])
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('5th element on 2nd row: ', arr[1, 4])
#Access 3d Array:
#Access the third element of the second array of the first array:
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr[0, 1, 2])
#Negative Indexing:
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('Last element from 2nd dim: ', arr[1, -1])
```

Code3:

```
#Slicing arrays:
#Slicing in python means taking elements from one given index to another given index.
#We pass slice instead of index like this: [start:end].
#We can also define the step, like this: [start:end:step]
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
#Slice elements from index 1 to index 5 from the following array:
print(arr[1:5])
```

```
#Slice elements from index 4 to the end of the array:
print(arr[4:])
#Slice elements from the beginning to index 4 (not included):
print(arr[:4])
#Negative Slicing:
#Slice from the index 3 from the end to index 1 from the end:
print(arr[-3:-1])
#STEP
#Use the step value to determine the step of the slicing:
#Return every other element from index 1 to index 5:
print(arr[1:5:2])
#Return every other element from the entire array:
print(arr[::2])
#Slicing 2-D Arrays
#From the second element, slice elements from index 1 to index 4 (not included):
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[1, 1:4])
#From both elements, return index 2:
print(arr[0:2, 2])
#From both elements, slice index 1 to index 4 (not included), this will return a 2-D array:
print(arr[0:2, 1:4])
```

Code4:

```
#Checking the Data Type of an Array
#The NumPy array object has a property called dtype that returns the data type of the array:
#Get the data type of an array object:
arr = np.array([1, 2, 3, 4])
print(arr.dtype)
#Get the data type of an array containing strings:
arr = np.array(['apple', 'banana', 'cherry'])
print(arr.dtype)
#Iterating Arrays
#Iterating means going through elements one by one.
#As we deal with multi-dimensional arrays in numpy, we can do this using basic for loop of python.
#If we iterate on a 1-D array it will go through each element one by one.
arr = np.array([1, 2, 3])
for x in arr:
  print(x)
#Iterate on each scalar element of the 2-D array:
arr = np.array([[1, 2, 3], [4, 5, 6]])
for x in arr:
  for y in x:
    print(y)
#Iterate on the elements of the following 3-D array:
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
for x in arr:
  print(x)
#To return the actual values, the scalars, we have to iterate the arrays in each dimension.
#Iterate down to the scalars:
for x in arr:
  for y in x:
    for z in y:
```

print(z)

2. Matplotlib

It is a very powerful plotting library useful for those working with Python and NumPy. The most used module of Matplotib is Pyplot which provides an interface like MATLAB but instead, it uses Python and it is open source

What is PIP

PIP is a package manager for Python packages, or modules if you like. Note: If you havePython version 3.4 or later, PIP is included by default.

Code1:

```
#Plotting x and y points
# The plot() function is used to draw points (markers) in a diagram.
# By default, the plot() function draws a line from point to point.
# The function takes parameters for specifying points in the diagram.
# Parameter 1 is an array containing the points on the x-axis.
# Parameter 2 is an array containing the points on the y-axis.
# If we need to plot a line from (1, 3) to (8, 10), we have to pass two arrays [1, 8] and [3, 10] to the plot function.
#Draw a line in a diagram from position (1, 3) to position (8, 10):

import matplotlib.pyplot as plt
import numpy as np
xpoints = np.array([1, 8])
ypoints = np.array([3, 10])
plt.plot(xpoints, ypoints)
plt.show()
```

Code2:

```
#Draw two points in the diagram, one at position (1, 3) and one in position (8, 10):
xpoints = np.array([1, 8])
ypoints = np.array([3, 10])
plt.plot(xpoints, ypoints, 'o')
plt.show()
#Draw a line in a diagram from position (1, 3) to (2, 8) then to (6, 1) and finally to position (8, 10):
xpoints = np.array([1, 2, 6, 8])
ypoints = np.array([3, 8, 1, 10])
plt.plot(xpoints, ypoints)
plt.show()
```

Code3:

```
#You can use also use the shortcut string notation parameter to specify the marker.
#This parameter is also called fmt, and is written with this syntax:
#marker|line|color
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, 'o:r')
plt.show()
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, linestyle = 'dotted')
plt.show()
```

Code4:

```
#Labels:
x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])
plt.plot(x, y)
plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")
plt.grid()
plt.show()
```

Code5:

```
#Subplots
# With the subplot() function you can draw multiple plots in one figure:
#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
plt.subplot(1, 2, 1)
plt.plot(x,y)
#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.show()
```

Code6:

```
#Bar plots
#With Pyplot, you can use the bar() function to draw bar graphs:
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])
plt.bar(x,y)
plt.show()
#for horizontal bar use 'barh'
plt.barh(x, y)
plt.show()
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])
plt.bar(x, y, color = "#4CAF50")
plt.show()
#histogram
x = np.random.normal(170, 10, 250)
plt.hist(x)
plt.show()
#Pie Chart
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
plt.pie(y, labels = mylabels, startangle = 90)
plt.show()
```

3. Pandas

Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data.
The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.
Pandas allows us to analyze big data and make conclusions based on statistical theories.
Pandas can clean messy data sets, and make them readable and relevant. Relevant data is very important in data science.

Pandas gives you answers about the data. Like:

Is there a correlation between two or more columns?
What is average value?
Max value?
Min value?
Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called cleaning the data.

```
#pip install pandas
```

Code1:

```
import pandas as pd
df = pd.read_csv("data.csv")
df.head()
print(df.shape)
print(df.columns)
print(df.info())
df.describe()
df["Pulse"].mean()
```

Code2:

```
import pandas as pd
mydataset = {
  'cars': ["BMW", "Volvo", "Ford"],
  'passings': [3, 7, 2]
}
myvar = pd.DataFrame(mydataset)
print(myvar)
```

Code3:

```
import pandas as pd
data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}
#load data into a DataFrame object:
df = pd.DataFrame(data)
```

```
print(df)
#refer to the row index:
print(df.loc[0])
#use a list of indexes:
print(df.loc[[0, 1]])
```

Code4:

```
#read CSV:
df = pd.read_csv('data.csv')
print(df)
#Analyzing dataframe:
#The head() method returns the headers and a specified number of rows, starting from the top.
df = pd.read_csv('data.csv')
#printing the first 10 rows of the DataFrame:
print(df.head(10))
#There is also a tail() method for viewing the last rows of the DataFrame.
#The tail() method returns the headers and a specified number of rows, starting from the bottom.
#Print the last 5 rows of the DataFrame:
print(df.tail())
#The DataFrames object has a method called info(), that gives you more information about the data set.
#Print information about the data:
print(df.info())
#Cleaning Empty cell:
new_df = df.dropna()
#If you want to change the original DataFrame, use the inplace = True argument:
#Remove all rows with NULL values:
df.dropna(inplace = True)
#The fillna() method allows us to replace empty cells with a value:
#Replace NULL values with the number 130:
df.fillna(130, inplace = True)
#Replace NULL values in the "Calories" columns with the number 130:
df["Calories"].fillna(130, inplace = True)
```

4. NLTK (Natural Language Toolkit)

This is one of the most usable and mother of all NLP libraries.
NLTK (Natural Language Toolkit) Library is a suite that contains libraries and programs for statistical language processing. It is one of the most powerful NLP libraries, which contains packages to make machines understand human language and reply to it with an appropriate response.
NLTK is used for tokenize words, Tokenization, Stemming, Lemmatization, Punctuation, Character count, word count, WordNet, Word Embedding, seq2seq model, etc.

In order to install NLTK run the following commands in your terminal.
sudo pip install nltk
Then, enter the python shell in your terminal by simply typing python
Type import nltk
nltk.download('all')
Some terms that will be frequently used are :
Corpus – Body of text, singular. Corpora is the plural of this.
Lexicon – Words and their meanings.
Token – Each "entity" that is a part of whatever was split up based on rules. For

examples, each word is a token when a sentence is “tokenized” into words. Each sentence can also be a token, if you tokenized the sentences out of a paragraph.
So basically tokenizing involves splitting sentences and words from the body of the text.

Code1:

```
import nltk
nltk.download('punkt')
```

Code2:

```
#Removing stop words with NLTK in Python
import nltk
nltk.download("stopwords")
from nltk.corpus import stopwords
print(stopwords.words('english'))
#The following program removes stop words from a piece of text:
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
example_sent = """This is a sample sentence,
showing off the stop words filtration."""
stop_words = set(stopwords.words('english'))
word_tokens = word_tokenize(example_sent)
# converts the words in word_tokens to lower case and then checks whether
#they are present in stop_words or not
filtered_sentence = [w for w in word_tokens if not w.lower() in stop_words]
#with no lower case conversion
filtered_sentence = []
for w in word_tokens:
if w not in stop_words:
filtered_sentence.append(w)
print(word_tokens)
print(filtered_sentence)
```

5. Spacy

Spacy is an open-source software python library used in advanced natural language processing and machine learning. It will be used to build information extraction, natural language understanding systems, and to pre-process text for deep learning.

Named Entity Recognition (NER) is a standard NLP problem which involves spotting named entities (people, places, organizations etc.) from a chunk of text, and classifying them into a predefined set of categories. Some of the practical applications of NER include:
Scanning news articles for the people, organizations and locations reported.
Providing concise features for search optimization: instead of searching the entire content, one may simply search for the major entities involved.
Quickly retrieving geographical locations talked about in Twitter posts.

SpaCy is regarded as the fastest NLP framework in Python, with single optimized functions for each of the NLP tasks it implements. Being easy to learn and use, one can easily perform simple tasks using a few lines of code.

Installation :

```
pip install spacy
```

```
python -m spacy download en_core_web_sm
```

Features Of Spacy:

- Non-destructive tokenization
- Named entity recognition
- Support for 61+ languages
- Part-of-speech tagging
- Built-in visualizers for syntax and NER
- Export to NumPy data arrays

Code1:

```
import spacy
nlp = spacy.load('en_core_web_sm')
sentence = "Apple is looking at buying U.K. startup for $1 billion"
doc = nlp(sentence)
for ent in doc.ents:
print(ent.text, ent.start_char, ent.end_char, ent.label_)
```

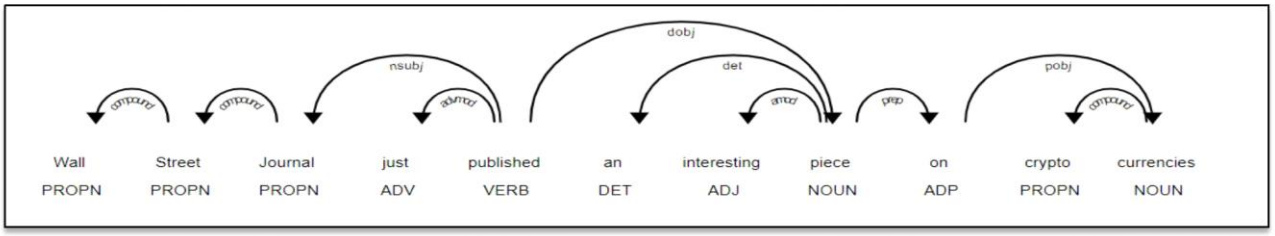
Code2:

```
# First we need to import spacy
import spacy
# Creating blank language object then
# tokenizing words of the sentence
nlp = spacy.blank("en")
doc = nlp("GeeksforGeeks is a one stop\
learning destination for geeks.")
for token in doc:
print(token)
```

Code3:

```
#Here is an example to show what other functionalities can be enhanced by adding modules to the
pipeline.
import spacy
# loading modules to the pipeline.
nlp = spacy.load("en_core_web_sm")
# Initialising doc with a sentence.
doc = nlp("If you want to be an excellent programmer \
, be consistent to practice daily on GFG.")
# Using properties of token i.e. Part of Speech and Lemmatization
for token in doc:
print(token, " | ",
spacy.explain(token.pos_),
" | ", token.lemma_)
```

```
#Code#18 Working with Tokens using loop in string
from spacy import displacy
doc = nlp('Wall Street Journal just published an interesting piece on cry
pto currencies')
displacy.render(doc, style='dep', jupyter=True, options={'distance': 90})
```



A modern syntactic dependency visualizer. Visualize spaCy's guess at the syntactic structure of a sentence. Arrows point from children to heads, and are labelled by their relation type.

NLTK is a string processing library. It takes strings as input and returns strings or lists of strings as output. Whereas, spaCy uses object-oriented approach. When we parse a text, spaCy returns document object whose words and sentences are objects themselves. spaCy has support for word vectors whereas NLTK does not.

6. Aima3 Library

aima3 stands for 'Artificial Intelligence: A Modern Approach'

Install pip install aima3

THE VACUUM WORLD

In this notebook, we will be discussing the structure of agents through an example of the vacuum agent. The job of AI is to design an agent program that implements the agent

function: the mapping from percepts to actions. We assume this program will run on some sort of computing device with physical sensors and actuators: we call this the architecture:

agent = architecture + program

Random Agent Program

A random agent program, as the name suggests, chooses an action at random, without taking into account the percepts.Here, we will demonstrate a random vacuum agent for a trivial vacuum environment, that is, the two-state environment. Let's begin by importing all the functions from the agents module:

```
from aima3.agents import *
from aima3.notebook import psource
#Let us first see how we define the TrivialVacuumEnvironment.
#Run the next cell to see how abstract class TrivialVacuumEnvironment is
defined in agents module.
```

```
class TrivialVacuumEnvironment(Environment):

"""This environment has two locations, A and B. Each can be Dirty or
Clean. The agent perceives its location and the location's status. This
```

```
serves as an example of how to implement a simple Environment."""

def init (self): super(). init ()
self.status = {loc_A: random.choice(['Clean', 'Dirty']), loc_B:
random.choice(['Clean', 'Dirty'])}

def thing_classes(self):
return [Wall, Dirt, ReflexVacuumAgent, RandomVacuumAgent,
TableDrivenVacuumAgent, ModelBasedVacuumAgent]

def percept(self, agent):
"""Returns the agent's location, and the location status
(Dirty/Clean)."""
return (agent.location, self.status[agent.location])

def execute_action(self, agent, action):
"""Change agent's location and/or location's status; track performance.
Score 10 for each dirt cleaned; -1 for each move."""
if action == 'Right': agent.location = loc_B agent.performance -= 1
elif action == 'Left': agent.location = loc_A

agent.performance -= 1 elif action == 'Suck':
if self.status[agent.location] == 'Dirty': agent.performance += 10
self.status[agent.location] = 'Clean'

def default_location(self, thing):
"""Agents start in either location at random."""
return random.choice([loc_A, loc_B]):
```

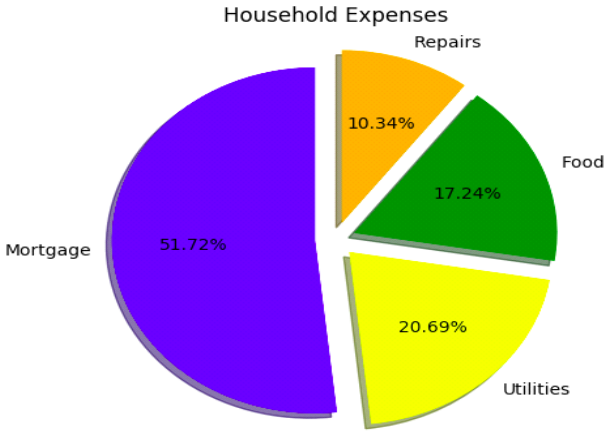
TASKS

- Use NumPy library and perform following tasks:

  1. Initialize 2 arrays and add their contents.
  2. Multiply all contents within one of the above arrays by an integer.
  3. Reshape one of the arrays to be 2D (if existing array is 2D, make it 3D).
  4. Convert One of the arrays to be of a different Data type.
  5. Generate a sequence of numbers in the form of a NumPy array from 0 to 100 with gaps of 2 numbers, for example: 0, 2, 4 ....
  6. From 2 NumPy arrays, extract the indexes in which the elements in the 2 arrays match.

- Use Matplotlib and perform following tasks:

  1. Create a line chart consisting of 2 separate lines each having a label of its own and one of them styled in a doted manner. Also add labels to the axes.
  2. Create a Pie Chart similar to the one given below. You can use dataset and colors of your own, but make sure that the structure is followed as is.



- Use pandas to create a new data frame consisting of 3 series and save it to a CSV file named 'TestSheet.csv'. Once created, retrieve data from the same file, make changes to it and add another series and save the new data frame to the existing file. See the sample below for how your data should look like.

	Duration	Pulse	Maxpulse	Calories
0	60	110	130	409.1
1	60	117	145	479
2	60	103	135	340
3	45	109	175	282.4
4	45	117	148	406

• Consider an *Interactive Cognitive Environment (ICE)* in which autonomous robot is performing cleaning task in the big room that appears to be a matrix of ***N * M***. Each index referred to as a cell of the matrix is valued as dirty "*D*" or clean "*C*". The cells which are occupied by the stuff in the room are blocked and valued "B". The vacuum can move in all four directions (up, down, left, right), and if the cell status is D, it will clean the cell and change the status to "C", if the cell status is either C, it will not enter the cell. The vacuum will stop working if the whole room is cleaned, i.e., the status of all the cells is either C. The vacuum may start cleaning the room from the first cell (0, 0) or any random location. You will trace the path of the vacuum and display at each step of the program. * Representthe location of the vacuum cleaner. Develop a Python code of the above describe scenario of the autonomous robot.

D*	D	D	D	D	D	D	
		C	D				
D	D	C	D		D	D	D
D	D	D	D	C	C	C	D
D	D	D	D	D	D	D	D
D	D	D	D	D	D	D	D
D	D	D			D	D	D

If vacuum is in a location where it's all neighbors (up, down, left and right) are clean (with status C) it will move in any one of the directions and keep searching the Dirt (D). It will stop it execution if it does not sense any dirt after 10 movements.

If vacuum is in a location where it's one more neighbor (up, down, left and right) is dirty it will move in any one of the directions and will return to the location when it cleans all the dirty cell of its neighbors. e.g., cell (0, 3) where it's three neighbors are dirty.

- Perform the following tasks with NLTK.

1. Write a Python NLTK program to split the text sentence/paragraph into a list of words. Sample Below.

```
Original string:
Joe waited for the train. The train was late. Mary and Samantha took the bus. I looked for Mary and Sama

Sentence-tokenized copy in a list:
['Joe waited for the train.', 'The train was late.', 'Mary and Samantha took the bus.', 'I looked for Ma

Read the list:
Joe waited for the train.
The train was late.
Mary and Samantha took the bus.
I looked for Mary and Samantha at the bus station.
```

2. Write a Python NLTK program to create a list of words from a given string.

```
Original string:
Joe waited for the train. The train was late. Mary and Samantha took the bus. I looked for Mary and Sama

List of words:
['Joe', 'waited', 'for', 'the', 'train', '.', 'The', 'train', 'was', 'late', '.', 'Mary', 'and', 'Samant
```

3. Write a Python NLTK program to tokenize words, sentence wise.

```
Original string:
Joe waited for the train. The train was late. Mary and Samantha took the bus. I looked for Mary and Sam

Tokenize words sentence wise:

Read the list:
['Joe', 'waited', 'for', 'the', 'train', '.']
['The', 'train', 'was', 'late', '.']
['Mary', 'and', 'Samantha', 'took', 'the', 'bus', '.']
['I', 'looked', 'for', 'Mary', 'and', 'Samantha', 'at', 'the', 'bus', 'station', '.']
```

- Use Spacy to perform the following tasks.
    1. Create a syntactic dependency visualizer for a given sentence.
    2. Break a given sentence into tokens each representing a single word.