1.

```
inputs = [1, 2, 3, 4]
weights = [[ 0.74864643, -1.00722027, 1.45983017, 1.34236011],
 [-1.20116017, -0.08884298, -0.46555646, 0.02341039],
 [-0.30973958, 0.89235565, -0.92841053, 0.12266543]]
biases = [0, 0.3, -0.5] # each neuron has a bias
layer_outputs = []
for neuron_weights, neuron_bias in zip(weights, biases):
 neuron_output = 0
 for n_input, weight in zip(inputs, neuron_weights):
 neuron_output += n_input * weight
 neuron_output += neuron_bias
 layer_outputs.append(neuron_output)
print(layer_outputs) # print the result
```

2.

a.

```
import numpy as np
import matplotlib.pyplot as plt
def ReLU(x):
 data = [max(0,value) for value in x]
 return np.array(data, dtype=float)
# Derivative for ReLU
def der_ReLU(x):
 data = [1 if value>0 else 0 for value in x]
 return np.array(data, dtype=float)
# Generating data for Graph
x_data = np.linspace(-10,10,100)
y_data = ReLU(x_data)
dy_data = der_ReLU(x_data)
# Graph
plt.plot(x_data, y_data, x_data, dy_data)
```

b.

```
import matplotlib.pyplot as plt
# Leaky Rectified Linear Unit (leaky ReLU) Activation Function
def leaky_ReLU(x):
 data = [max(0.05*value,value) for value in x]
 return np.array(data, dtype=float)
# Derivative for leaky ReLU
def der_leaky_ReLU(x):
 data = [1 if value>0 else 0.05 for value in x]
 return np.array(data, dtype=float)
# Generating data For Graph
x_data = np.linspace(-10,10,100)
y_data = leaky_ReLU(x_data)
dy_data = der_leaky_ReLU(x_data)
# Graph
plt.plot(x_data, y_data, x_data, dy_data)
plt.title('leaky ReLU Activation Function & Derivative')
plt.legend(['leaky_ReLU','der_leaky_ReLU'])
plt.grid()
plt.show()
```

c.

```
import numpy as np
from matplotlib import pyplot as plt
# Sigmoidal
def sig(x):
 return 1/(1+np.exp(-x))
```

```
# Sigmoidal derivative
def dsig(x):
 return sig(x) * (1- sig(x))
# Generating data to plot
x_data = np.linspace(-6,6,100)
y_data = sig(x_data)
dy_data = dsig(x_data)
# Plotting
plt.plot(x_data, y_data, x_data, dy_data)
plt.title('Sigmoid Function & Derivative')
plt.legend(['f(x)','f\'(x)'])
plt.grid()
plt.show()
```

d.

```
import numpy as np
import matplotlib.pyplot as plt
# Hyperbolic Tangent (htan) Activation Function
def htan(x):
 return (np.exp(x) - np.exp(-x))/(np.exp(x) + np.exp(-x))
# htan derivative
def der_htan(x):
 return 1 - htan(x) * htan(x)
# Generating data for Graph
x_data = np.linspace(-6,6,100)
y_data = htan(x_data)
dy_data = der_htan(x_data)
# Graph
plt.plot(x_data, y_data, x_data, dy_data)
plt.title('htan Activation Function & Derivative')
plt.legend(['htan','der_htan'])
plt.grid()
plt.show()
```

3.

```
importing Python library
import numpy as np
def unitStep(v):
    if v >= 0:
        return 1
    else:
        return 0
def perceptronModel(x, w, b):
    v = np.dot(w, x) + b
    y = unitStep(v)
    return y
def NOT_logicFunction(x):
    wNOT = -1
    bNOT = 0.5
    return perceptronModel(x, wNOT, bNOT)
def AND_logicFunction(x):
    w = np.array([1, 1])
    bAND = -1.5
    return perceptronModel(x, w, bAND)
def OR_logicFunction(x):
    w = np.array([1, 1])
    bOR = -0.5
    return perceptronModel(x, w, bOR)
def XOR_logicFunction(x):
    y1 = AND_logicFunction(x)
    y2 = OR_logicFunction(x)
    y3 = NOT_logicFunction(y1)
    final_x = np.array([y2, y3])
    finalOutput = AND_logicFunction(final_x)
    return finalOutput

test1 = np.array([0, 1])
test2 = np.array([1, 1])
test3 = np.array([0, 0])
test4 = np.array([1, 0])

print("XOR({}, {}) = {}".format(0, 1, XOR_logicFunction(test1)))
print("XOR({}, {}) = {}".format(1, 1, XOR_logicFunction(test2)))
print("XOR({}, {}) = {}".format(0, 0, XOR_logicFunction(test3)))
print("XOR({}, {}) = {}".format(1, 0, XOR_logicFunction(test4)))
```

4.

```
from sklearn import datasets
import pandas as pd□□□
import numpy as np

iris = datasets.load_iris() #Loading the dataset
print(iris.keys())
iris = pd.DataFrame(
    data= np.c_[iris['data'], iris['target']],
    columns= iris['feature_names'] + ['target']
    )
print(iris.head(10))
species = []

for i in range(len(iris['target'])):
    if iris['target'][i] == 0:
        species.append("setosa")
    elif iris['target'][i] == 1:
        species.append('versicolor')
    else:
        species.append('virginica')
iris['species'] = species
iris.groupby('species').size()
print(species)

import matplotlib.pyplot as plt

setosa = iris[iris.species == "setosa"]
versicolor = iris[iris.species=='versicolor']
virginica = iris[iris.species=='virginica']

fig, ax = plt.subplots()
fig.set_size_inches(13, 7) # adjusting the length and width of plot

# lables and scatter points
ax.scatter(setosa['petal length (cm)'], setosa['petal width (cm)'], label="Setosa", facecolor="blue")
ax.scatter(versicolor['petal length (cm)'], versicolor['petal width (cm)'], label="Versicolor", facecolor="green")
ax.scatter(virginica['petal length (cm)'], virginica['petal width (cm)'], label="Virginica", facecolor="red")

ax.set_xlabel("petal length (cm)")
ax.set_ylabel("petal width (cm)")
ax.grid()
ax.set_title("Iris petals")
ax.legend()
```