

**Final Year B. Tech (EE)**

**Trimester: I**

**Subject:**

**Artificial Intelligence and Machine Learning**

**Name: Shreerang Mhatre**

**Roll No: 52**

**Class: TY**

**Batch: A3**

**Experiment No: 01**

**Name of the Experiment**:Calculate the output of a simple neuron



**Marks** **Teacher’s Signature with date**

**Performed on: 14/09/2023**

**Submitted on: 29/09/2023**



**Aim:** To create a simple neural network and calculate its output.

**Prerequisite:** Knowledge of logic gates, perceptron, various activation functions.

**Objective:**

To create a simple single layer neural network and calculate its output using Python Programming.

**Components and Equipment required:**

**Python software**

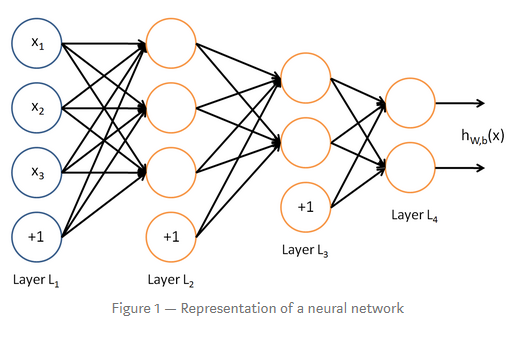
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | |  | **Expt. 1- 1** | |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  | **WWW.MITWPU.COM** | | |



**Theory:**

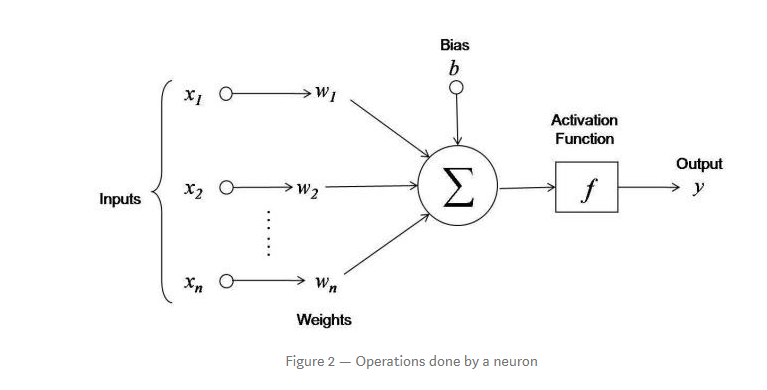
Based on nature, neural networks are the usual representation we make of the brain: neurons interconnected to other neurons which forms a network. The operation of a complete neural network is straightforward: one enters variables as inputs, and after some calculations, an output is returned. Artificial neural network is usually put on columns, so that a neuron of the column *n* can only be connected to neurons from columns *n-1*and*n+1.*There are few types of networks that use a different architecture.

A simple artificial neural network is represented as below:



Neural networks can usually be read from left to right. Here, the first layer is the layer in which inputs are entered. There are 2 internals layers (called hidden layers) that do some math, and one last layer that contains all the possible outputs. “+1” s at the bottom of every column, it is something called “bias”.

O**peration of a neuron:**



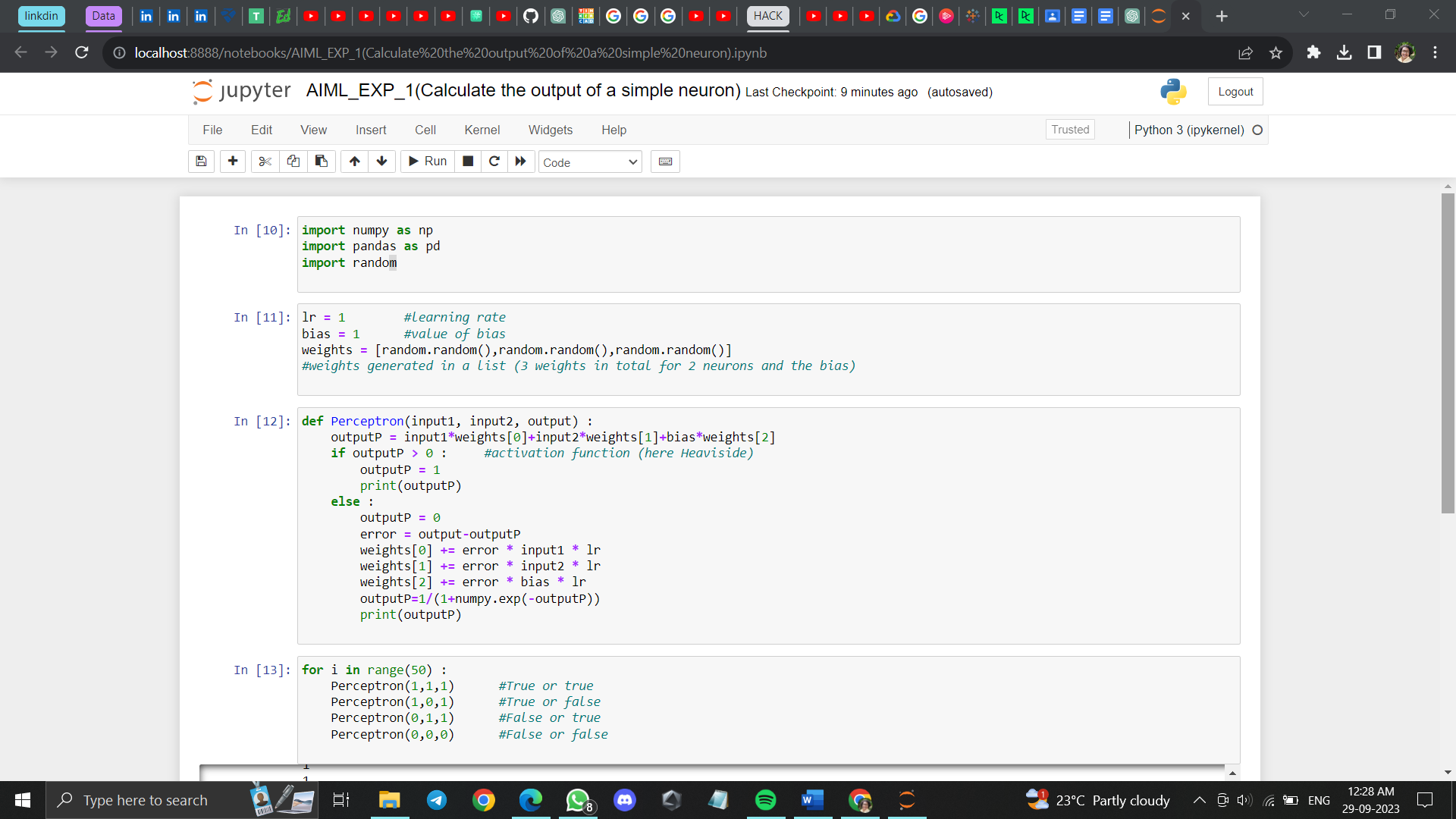
|  |  |  |
| --- | --- | --- |
|  |  | |
|  |  | |
| First, it adds up the value of every neuron from the previous column it is connected to. On the Figure 2, there are 3 inputs (x1, x2, x3) coming to the neuron, so 3 neurons of the previous column are connected to our neuron. This value is multiplied, before being added, by another variable called “weight” (w1, w2, w3) which determines the connection between the two neurons. Each connection of neurons has its own weight, and those are the only values that will be modified during the learning process.  Moreover, a bias value may be added to the total value calculated. It is not a value coming from a specific neuron and is chosen before the learning phase, but can be useful for the network.  After all those summations, the neuron finally applies a function called “activation function” to the obtained value. Thus a neuron takes all values from connected neurons multiplied by their respective weight, add them, and apply an activation function. Then, the neuron is ready to send its new value to other neurons.  After every neurons of a column did it, the neural network passes to the next column. In the end, the last values obtained should be one usable to determine the desired output.  **Network under consideration:**  It consists of 2 neurons in the inputs column and 1 neuron in the output column. This configuration allows to create a simple classifier to distinguish 2 groups.    **To create a simple Neural Network using Python:**  **Step-1**  **import** numpy, random, os lr = 1 #learning rate bias = 1 #value of bias weights = [random.random(),random.random(),random.random()] #weights generated in a list (3 weights in total for 2 neurons and the bias)  In the beginning of the program we define libraries and the values of the parameters, and creates a list which contains the values of the weights that will be modified.  Step-2  **def** Perceptron(input1, input2, output) : outputP = input1\*weights[0]+input2\*weights[1]+bias\*weights[2] **if** outputP > 0 : #activation function (here Heaviside) outputP = 1 **else** : outputP = 0 error = output – outputP weights[0] += error \* input1 \* lr weights[1] += error \* input2 \* lr weights[2] += error \* bias \* lr  Here we create a function which defines the work of the output neuron. It takes 3 parameters (the 2 values of the neurons and the expected output). “outputP” is the variable corresponding to the output given by the Perceptron. Then we calculate the error, used to modify the weights of every connection to the output neuron right after.  Step-3  **for** i in range(50) : Perceptron(1,1,1) #True or true Perceptron(1,0,1) #True or false Perceptron(0,1,1) #False or true Perceptron(0,0,0) #False or false  We create a loop that makes the neural network repeat every situation several times. This part is the learning phase. The number of iteration is chosen according to the precision we want.  **Conclusion:**  **Post Lab Questions:**   1. Comment on the similarities between biological neuron and artificial   neuron.   1. What do you mean by activation function? State and explain its types. 2. Implement the above code considering sigmoid function.   (outputP = 1/(1+numpy.exp(-outputP)) #sigmoid function) | |  |
|  |  | |
|  |  | |
|  |  | |

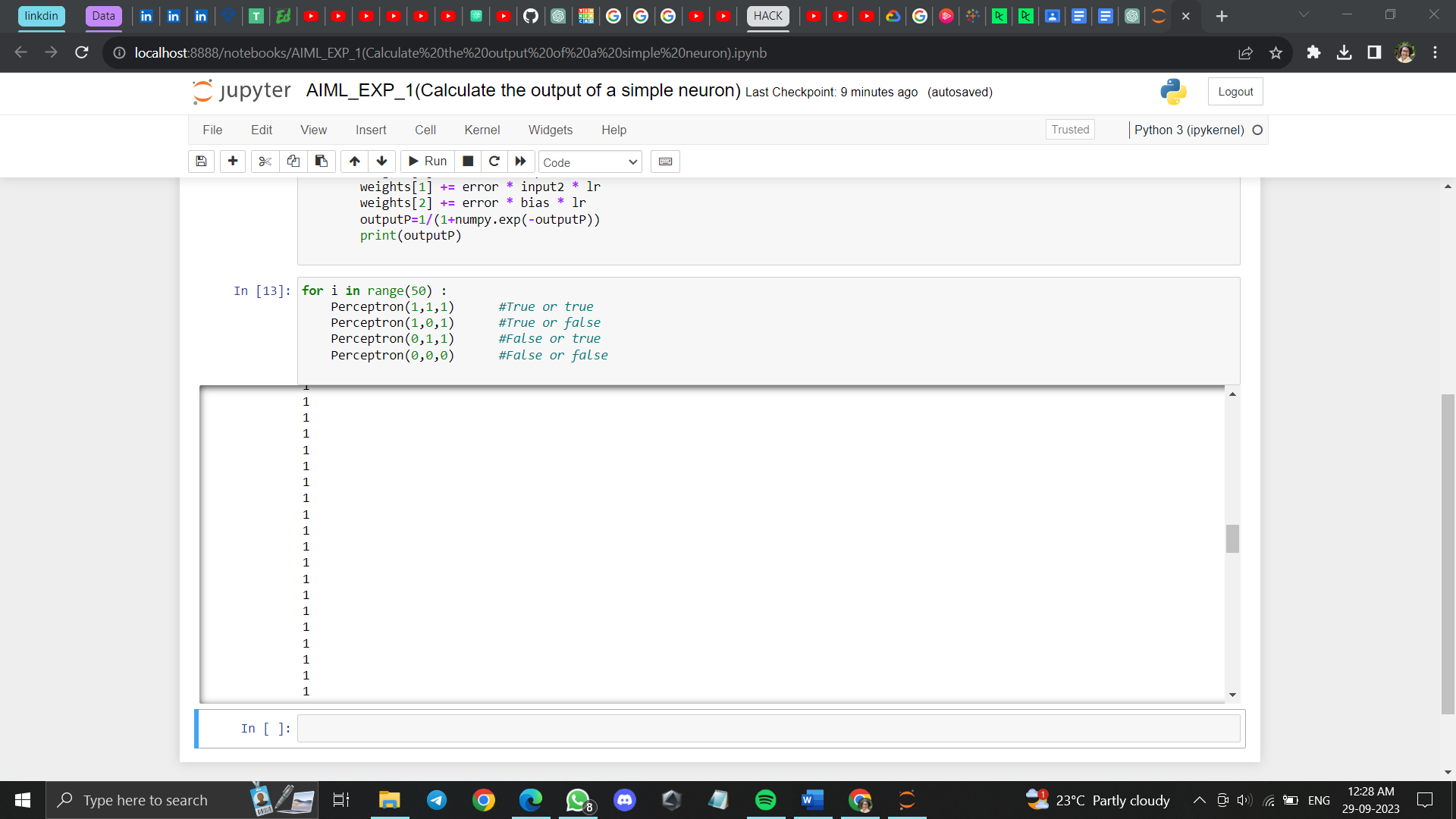
**Expt. 1- 2**

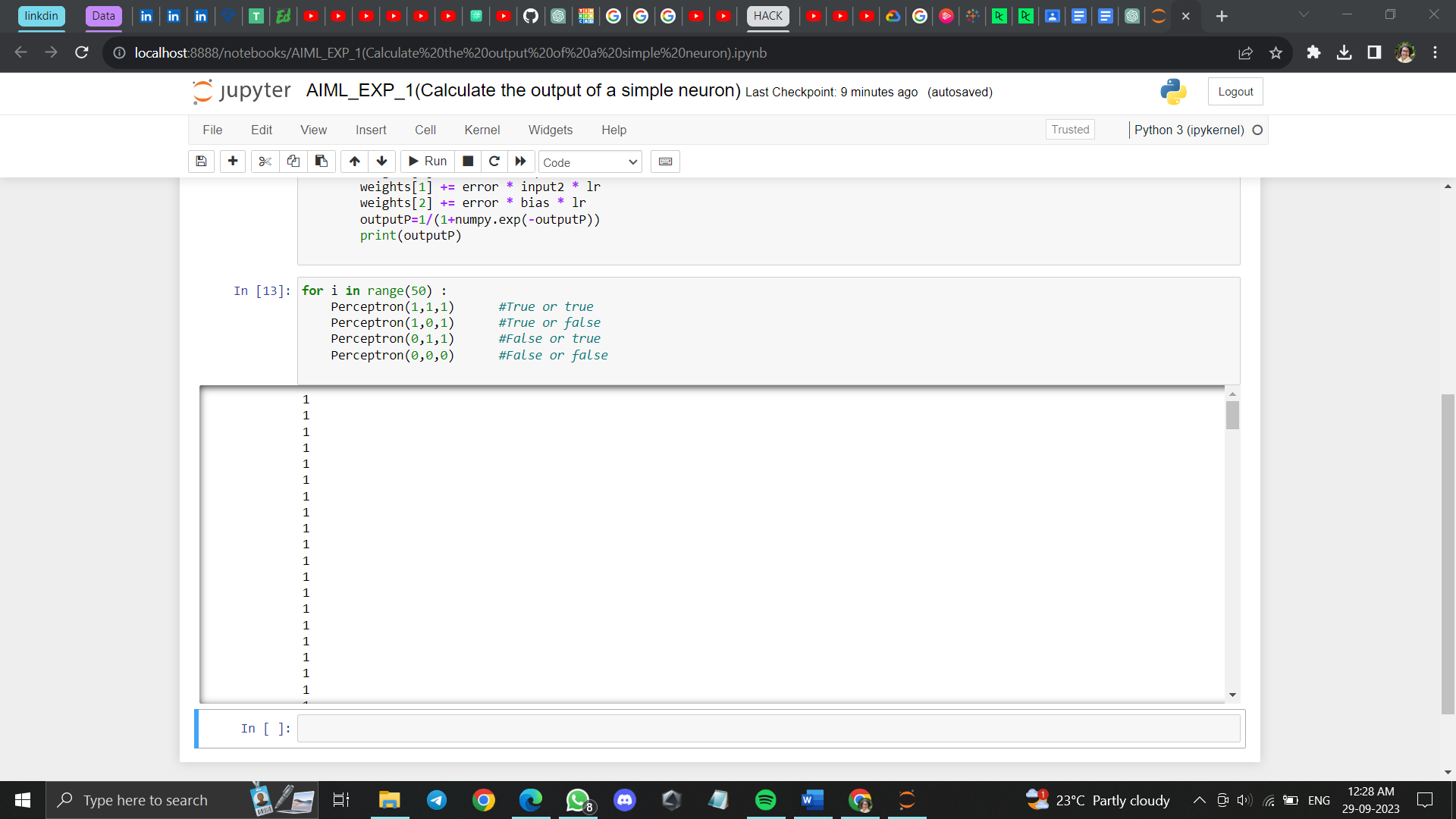
**WWW.MITWPU.COM**



.







|  |  |
| --- | --- |
|  |  |

**Expt. 1- 3**

**WWW.MITWPU.C**

