Problem 1a: Handling Missing Values in a Dataset

You are given a CSV file containing Titanic passenger data. Some passengers have missing age values. Write a Python program using NumPy to handle missing values as follows:

1. Convert the "Sex" column to numerical values (0 for male, 1 for female).
2. Calculate the mean age separately for males and females (skipping the missing values).
3. Replace missing age values with the mean age of the respective gender.
4. Print the mean male and female age, and the age assigned to the 41st passenger, Dr. Arthur Jackson Brewe.

Problem 1b: Class-Based Mean Fare Calculation

The Titanic dataset categorizes passengers into three classes (column “pclass”). Write a Python program using NumPy that:

1. Computes the mean ticket price for each class (1st, 2nd, and 3rd) (skipping the missing values).
2. If a ticket price is missing, replace it with the mean fare of the respective class.
3. Print the first, second and the third class mean fare values.
4. Print the fare value assigned to the 1226 th passenger Mr. Thomas Storey.

Problem 1c: Z-Score Normalization for Ticket Prices and Age

Z-score normalization is a crucial step in machine learning to standardize numerical data. Implement a function “zscore” that:

1. Takes a list of numerical values as input.
2. Calculates the mean and standard deviation of the list.
3. Returns a list where each value is transformed using the Z-score formula: $Z = (x - \mu)/\sigma$
4. Apply this function to the ticket prices and age separately in the Titanic dataset. Print the normalized ticket price and age values for the first passenger.

Problem 1d: Preparing the dataset

Using the processed Titanic dataset, build a dataset which can be readily used for training a classifier.

1. Use the following as features (X): Passenger class, sex, age (Z-score normalized), and fare (Z-score normalized).
2. Use the “survived” column values (0 or 1) as the expected output (y)

3. Split the dataset into **training: 80% (X_train and y_train)** and **testing: 20% (X_test and y_test)** sets. You should use np.random for randomly selecting the samples for the training and testing set.
4. Print the shapes of the X_train, X_test, y_train and y_test arrays.

Bonus Problem 1e: Logistic Regression for Titanic Survival Prediction

Using the built dataset, train a logistic regression model to predict whether a passenger survived or not.

1. Train a logistic regression model. Evaluate the model's performance using **accuracy** as the metric.
2. Use scikit-learn library for training the model and computing the test accuracy
3. Print the test accuracy.

Problem 2: Image Processing by Convolution

Convolution is a fundamental operation in image processing and machine learning, particularly in convolutional neural networks (CNNs). It helps extract features like edges, textures, and patterns from images. Conceptually, convolution can be thought of as a **dot product operation** between a small matrix (called a kernel or filter) and overlapping sections of the input image.

How Convolution Works

1. **Sliding the Kernel Over the Image:**
   A kernel (typically a small 3×3 or 5×5 matrix) moves across the image, performing element-wise multiplication with the corresponding image pixels.
2. **Dot Product Computation:**
   The values of the kernel are multiplied with the corresponding pixel values in the image, and the results are summed up to form a single output pixel.
3. **Building the Output Image:**
   This process is repeated across the entire image, creating a new matrix that represents a transformed version of the original image.

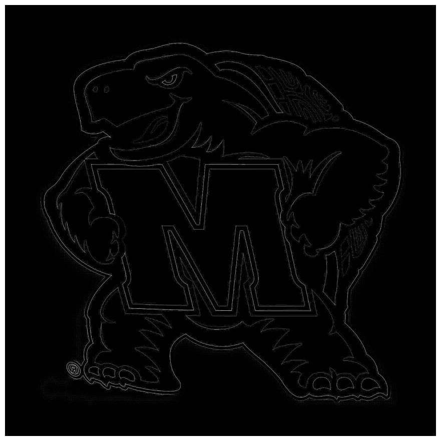
Implement a simple **2D convolution operation** using NumPy for edge detection. The kernel used for edge detection is:

-1	-1	-1
-1	8	-1
-1	-1	-1

1. Read the “Testudo.jpg” image as a numpy array using cv2 and numpy library.
2. Perform convolution using the above filter. Assume stride of 1 and padding 1 (use numpy.pad)
3. Flip the convolution output using the numpy.flip function.
4. Save the flipped output as “Testudo_out.jpg”.

The images below represent the input and the expected output:

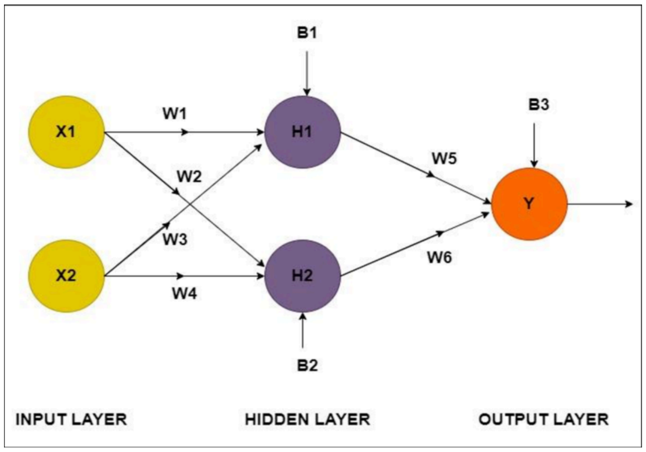
Input Image


processed output image

Problem 3: Solving XOR with a Deep Neural Network

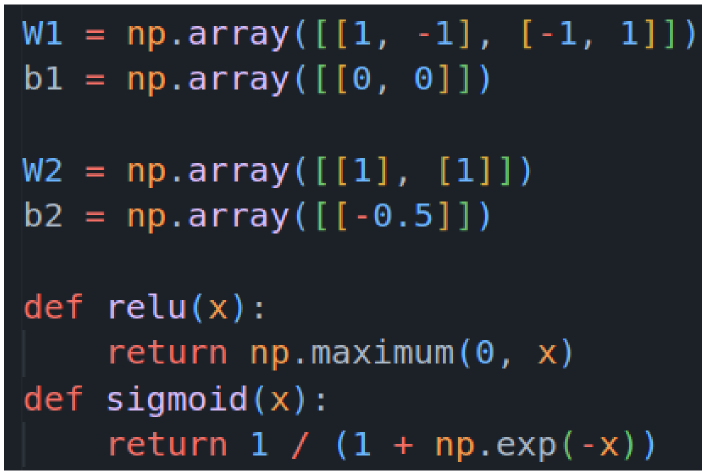
The XOR problem is a fundamental example of a problem that linear classifiers cannot solve. Implement a 2-layer perceptron to solve XOR using numpy.

1. The code should take inputs of first and second bit values and print the corresponding XOR output.
2. Build a neural network with **a single hidden layer**. The network architecture is as given below:



(Image taken from google)

- The hidden layer has 2 neurons and a single output is expected.
- Use the weight and bias values given below and implement linear operation to get the output.
- The linear operation is $z = w.x + b$ where the x is the input/previous layer value, w is the weight and b is the bias.
- Apply relu function at the output of the hidden layer and sigmoid function at the output of the output layer.



```
W1 = np.array([[1, -1], [-1, 1]])
b1 = np.array([[0, 0]])

W2 = np.array([[1], [1]])
b2 = np.array([[-0.5]])

def relu(x):
    return np.maximum(0, x)
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

What to Submit:

Submit a Jupyter Notebook file named <First Name>_<Last Name>_hw2.ipynb. Ensure all code is **well-commented**, and all required outputs are displayed. The completed assignment should be uploaded to Canvas. Please do not use any other library except numpy, cv2 (for problem 2), scikit-learn (for bonus problem) and csv (for reading the csv file). Also submit the "Testudo_out.jpg" image generated as part of problem 2.