```
import numpy as np
import pandas as pd

input_value = np.array([[0, 0], [0, 1], [1, 1], [1, 0]])
print(input_value.shape)  # (4, 2)

output = np.array([0, 0, 1, 0])
output = output.reshape(4, 1)
print(output.shape)  # (4, 1)
```

```
(4, 2)
(4, 1)
```

```
weights = np.array([[0.1], [0.3]])
print(weights)  # array([[0.1], [0.3]])
bias = 0.2

def sigmoid_func(x):
    return 1 / (1 + np.exp(-x))

def der(x):
    return sigmoid_func(x) * (1 - sigmoid_func(x))
```

```
[[0.1]
 [0.3]]
```

```
for epochs in range(15000):
    input_arr = input_value
    weighted_sum = np.dot(input_arr, weights) + bias
    first_output = sigmoid_func(weighted_sum)

    # Error and derivative calculation
    error = first_output - output
    total_error = np.square(np.subtract(first_output, output)).mean()

    first_der = error
    second_der = der(first_output)
    derivative = first_der * second_der

    t_input = input_value.T
    final_derivative = np.dot(t_input, derivative)

    # Gradient descent update
    weights = weights - (0.05 * final_derivative)
    for i in derivative:
        bias = bias - (0.05 * i)

print("Final weights:\n", weights)
print("Final bias:\n", bias)
```

```
Final weights:
 [[6.62916366]
 [6.62916441]]
Final bias:
 [-10.23197316]
```

```
preds = [
    np.array([1, 0]),
    np.array([1, 1]),
    np.array([0, 0]),
    np.array([0, 1])
]

for pred in preds:
    result = np.dot(pred, weights) + bias
    res = sigmoid_func(result)
    print(f"Input: {pred}, Output: {res}")
```

```
Input: [1 0], Output: [0.02652435]
Input: [1 1], Output: [0.95375065]
Input: [0 0], Output: [3.59993686e-05]
```

```
Input: [0 1], Output: [0.02652437]
```

```
predicted_prob = first_output        # sigmoid probabilities
predicted_class = (predicted_prob >= 0.5).astype(int)  # threshold

# Flatten for sklearn metrics
y_true = output.flatten()
y_pred = predicted_class.flatten()

# Metrics
accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred, zero_division=0)
recall = recall_score(y_true, y_pred, zero_division=0)
f1 = f1_score(y_true, y_pred, zero_division=0)
cm = confusion_matrix(y_true, y_pred)

print("\n=== PERFORMANCE METRICS ===")
print(f"Accuracy  : {accuracy:.4f}")
print(f"Precision : {precision:.4f}")
print(f"Recall    : {recall:.4f}")
print(f"F1-Score  : {f1:.4f}")
print("\nConfusion Matrix:\n", cm)
```

```
=== PERFORMANCE METRICS ===
Accuracy  : 1.0000
Precision : 1.0000
Recall    : 1.0000
F1-Score  : 1.0000

Confusion Matrix:
 [[3 0]
 [0 1]]
```