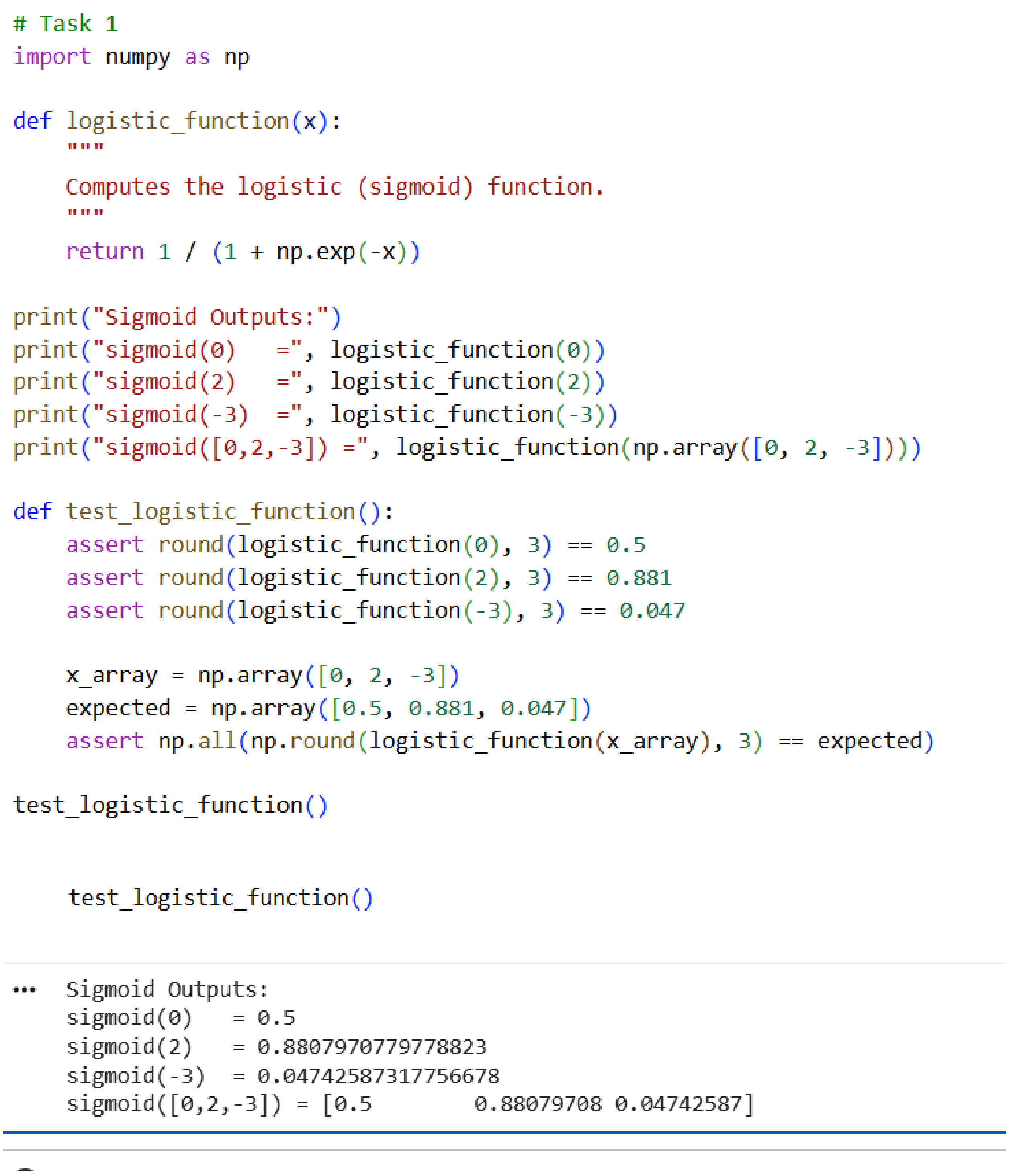
Result of Worksheet6



```
# Task 1
import numpy as np

def logistic_function(x):
    """
    Computes the logistic (sigmoid) function.
    """
    return 1 / (1 + np.exp(-x))

print("Sigmoid Outputs:")
print("sigmoid(0)   =", logistic_function(0))
print("sigmoid(2)   =", logistic_function(2))
print("sigmoid(-3)  =", logistic_function(-3))
print("sigmoid([0,2,-3]) =", logistic_function(np.array([0, 2, -3])))

def test_logistic_function():
    assert round(logistic_function(0), 3) == 0.5
    assert round(logistic_function(2), 3) == 0.881
    assert round(logistic_function(-3), 3) == 0.047

    x_array = np.array([0, 2, -3])
    expected = np.array([0.5, 0.881, 0.047])
    assert np.all(np.round(logistic_function(x_array), 3) == expected)

test_logistic_function()


    test_logistic_function()
```

```
Sigmoid Outputs:
sigmoid(0)   = 0.5
sigmoid(2)   = 0.8807970779778823
sigmoid(-3)  = 0.04742587317756678
sigmoid([0,2,-3]) = [0.5        0.88079708 0.04742587]
```

```
# Task 2: Log Loss
def log_loss(y_true, y_pred):
    y_pred = np.clip(y_pred, 1e-10, 1 - 1e-10)
    return -(y_true*np.log(y_pred) + (1-y_true)*np.log(1-y_pred))

print("Log Loss Outputs:")
print("log_loss(1, 0.9) =", log_loss(1, 0.9))
print("log_loss(0, 0.1) =", log_loss(0, 0.1))

assert np.isclose(log_loss(1,1), 0.0)
assert np.isclose(log_loss(0,0), 0.0)
assert np.isclose(log_loss(1,0.8), -np.log(0.8))
```

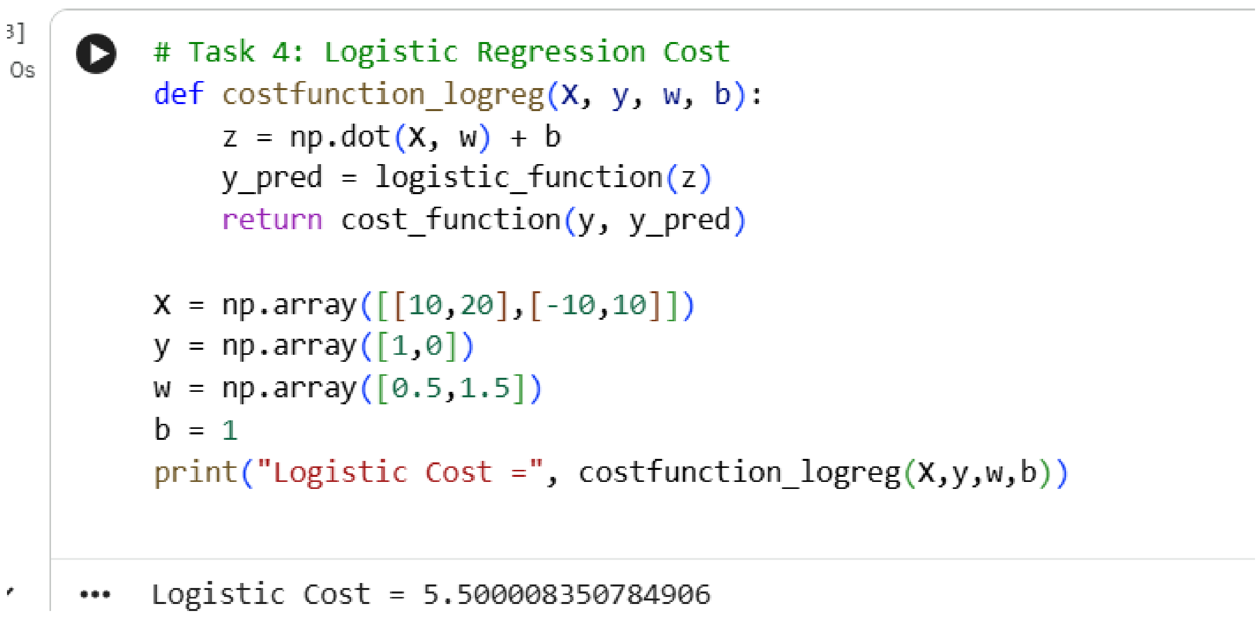
```
Log Loss Outputs:
log_loss(1, 0.9) = 0.10536051565782628
log_loss(0, 0.1) = 0.10536051565782628
```

```
# Task 3: Cost Function
def cost_function(y_true, y_pred):
    n = len(y_true)
    loss = -(y_true*np.log(y_pred) + (1-y_true)*np.log(1-y_pred))
    return np.sum(loss) / n

y_true = np.array([1,0,1])
y_pred = np.array([0.9,0.1,0.8])
print("Cost =", cost_function(y_true, y_pred))

expected = ( -np.log(0.9) - np.log(0.9) - np.log(0.8) ) / 3
assert np.isclose(cost_function(y_true, y_pred), expected)
```

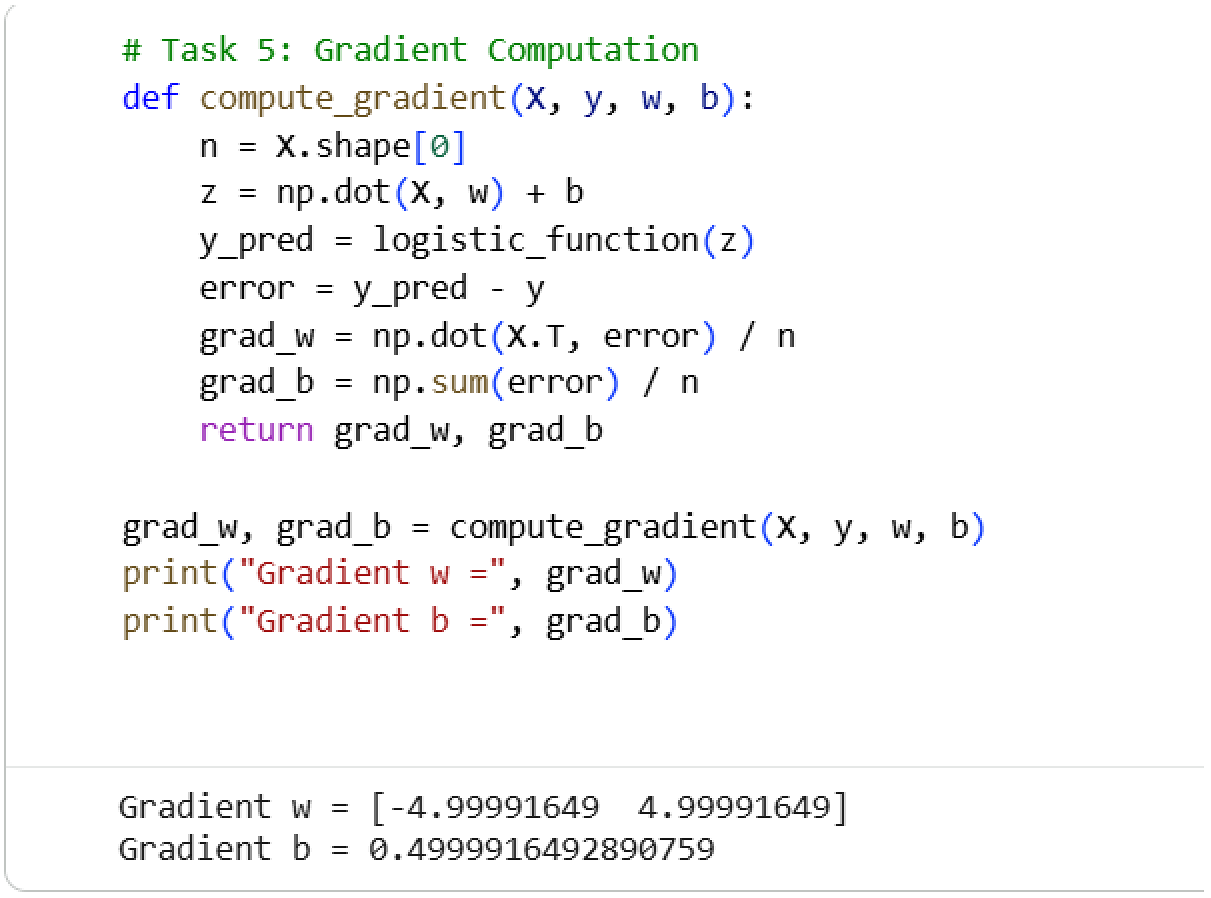
```
Cost = 0.14462152754328741
```



```python
# Task 4: Logistic Regression Cost
def costfunction_logreg(X, y, w, b):
    z = np.dot(X, w) + b
    y_pred = logistic_function(z)
    return cost_function(y, y_pred)

X = np.array([[10,20],[-10,10]])
y = np.array([1,0])
w = np.array([0.5,1.5])
b = 1
print("Logistic Cost =", costfunction_logreg(X,y,w,b))
```

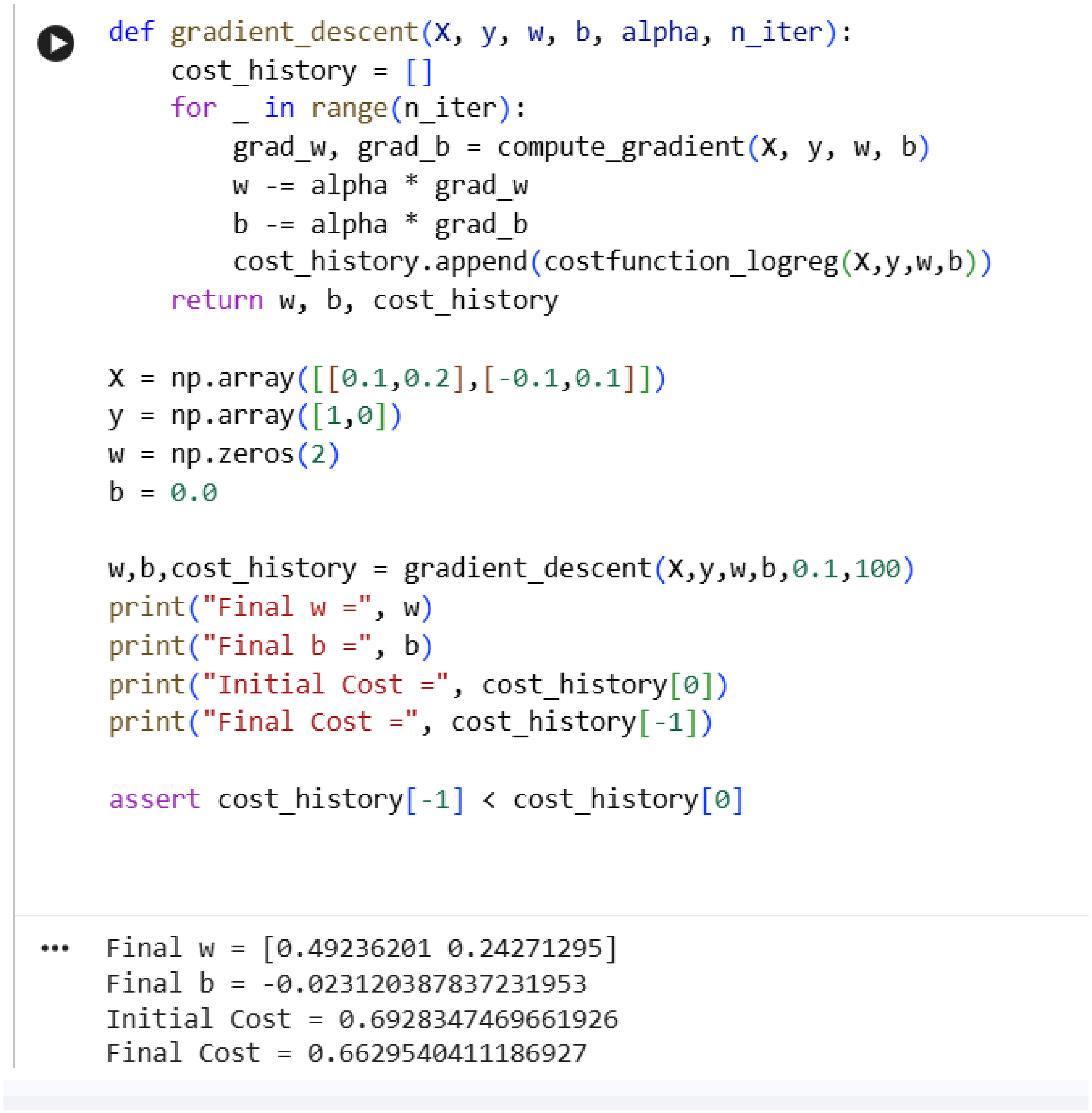
```
Logistic Cost = 5.500008350784906
```



```python
# Task 5: Gradient Computation
def compute_gradient(X, y, w, b):
    n = X.shape[0]
    z = np.dot(X, w) + b
    y_pred = logistic_function(z)
    error = y_pred - y
    grad_w = np.dot(X.T, error) / n
    grad_b = np.sum(error) / n
    return grad_w, grad_b

grad_w, grad_b = compute_gradient(X, y, w, b)
print("Gradient w =", grad_w)
print("Gradient b =", grad_b)
```

```
Gradient w = [-4.99991649  4.99991649]
Gradient b = 0.4999916492890759
```



```python
def gradient_descent(X, y, w, b, alpha, n_iter):
    cost_history = []
    for _ in range(n_iter):
        grad_w, grad_b = compute_gradient(X, y, w, b)
        w -= alpha * grad_w
        b -= alpha * grad_b
        cost_history.append(costfunction_logreg(X,y,w,b))
    return w, b, cost_history

X = np.array([[0.1,0.2],[-0.1,0.1]])
y = np.array([1,0])
w = np.zeros(2)
b = 0.0

w,b,cost_history = gradient_descent(X,y,w,b,0.1,100)
print("Final w =", w)
print("Final b =", b)
print("Initial Cost =", cost_history[0])
print("Final Cost =", cost_history[-1])

assert cost_history[-1] < cost_history[0]
```

```
Final w = [0.49236201 0.24271295]
Final b = -0.023120387837231953
Initial Cost = 0.6928347469661926
Final Cost = 0.6629540411186927
```

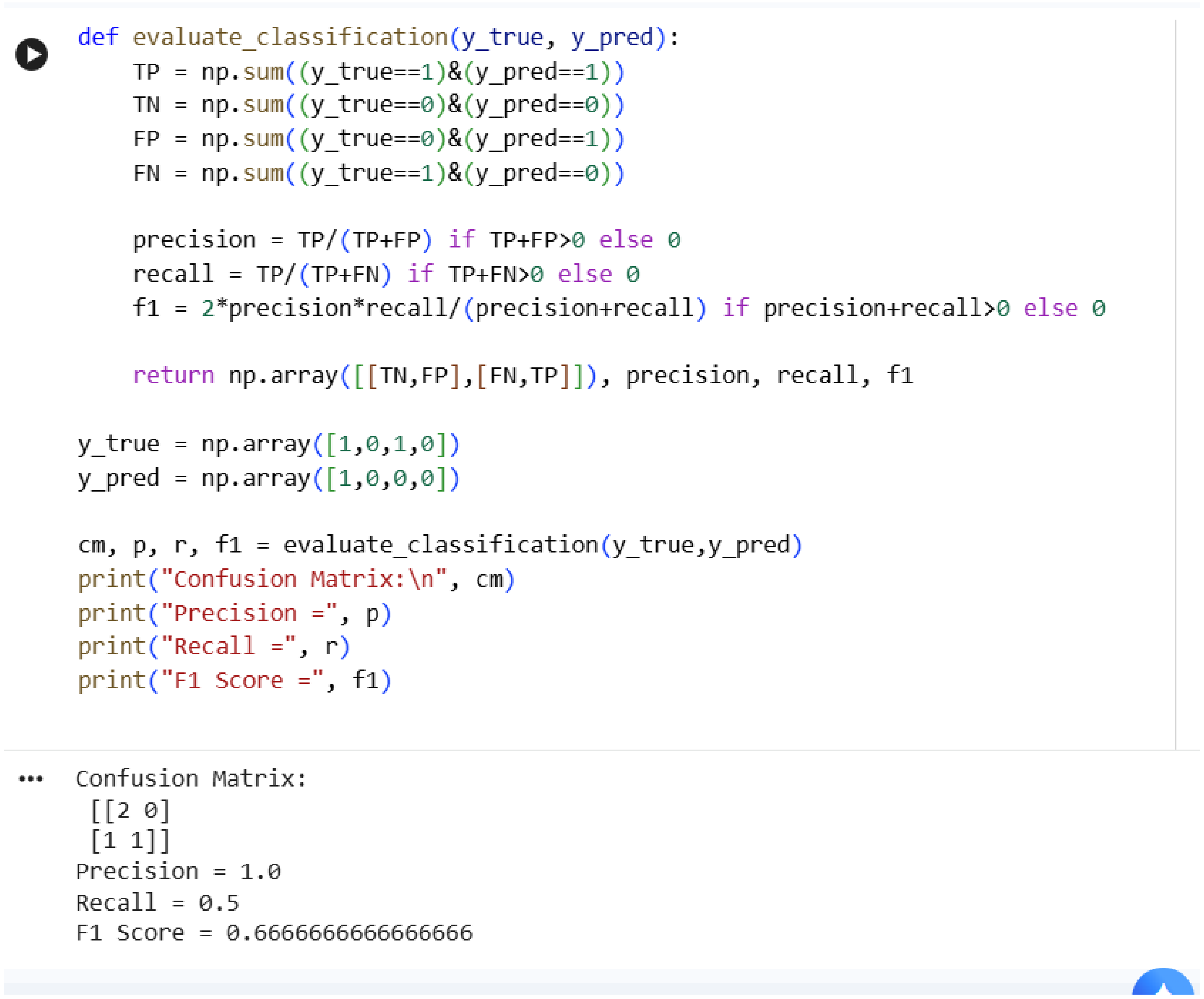
```python
# Task 7: Prediction
def prediction(X, w, b, threshold=0.5):
    z = np.dot(X, w) + b
    probs = logistic_function(z)
    return (probs >= threshold).astype(int)

X_test = np.array([[0.5,1.0],[1.5,-0.5],[-0.5,-1.0]])
w_test = np.array([1.0,-1.0])
b_test = 0.0

y_pred = prediction(X_test,w_test,b_test)
print("Predictions =", y_pred)

assert np.array_equal(y_pred, np.array([0,1,1]))
```

```
Predictions = [0 1 1]
```



```python
def evaluate_classification(y_true, y_pred):
    TP = np.sum((y_true==1)&(y_pred==1))
    TN = np.sum((y_true==0)&(y_pred==0))
    FP = np.sum((y_true==0)&(y_pred==1))
    FN = np.sum((y_true==1)&(y_pred==0))

    precision = TP/(TP+FP) if TP+FP>0 else 0
    recall = TP/(TP+FN) if TP+FN>0 else 0
    f1 = 2*precision*recall/(precision+recall) if precision+recall>0 else 0

    return np.array([[TN,FP],[FN,TP]]), precision, recall, f1

y_true = np.array([1,0,1,0])
y_pred = np.array([1,0,0,0])

cm, p, r, f1 = evaluate_classification(y_true,y_pred)
print("Confusion Matrix:\n", cm)
print("Precision =", p)
print("Recall =", r)
print("F1 Score =", f1)
```

```
Confusion Matrix:
 [[2 0]
 [1 1]]
Precision = 1.0
Recall = 0.5
F1 Score = 0.6666666666666666
```