```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers


# Define the function to be approximated
def my_func(x):
    return np.sin(x)


# Define the number of terms in the Taylor expansion
n_terms = 5


# Define the input range for the function
x_min, x_max = 0, 2*np.pi


# Define the number of training examples
n_examples = 1000


# Generate the training data
x_train = np.random.uniform(x_min, x_max, size=(n_examples,))
y_train = np.zeros((n_examples, n_terms))

for i in range(n_terms):
    y_train[:, i] = ((-1)**i / np.math.factorial(2*i + 1)) * np.power(x_train, 2*i + 1)


# Define the Transformer model architecture
inputs = keras.Input(shape=(1,))
x = layers.Dense(32, activation="relu")(inputs)
x = layers.Dense(32, activation="relu")(x)
x = layers.Dense(32, activation="relu")(x)
outputs = layers.Dense(n_terms)(x)

model = keras.Model(inputs=inputs, outputs=outputs)


# Compile the model
model.compile(loss="mse", optimizer="adam")


# Train the model
history = model.fit(x_train, y_train, epochs=100, batch_size=32, verbose=1)
```

```
32/32 [==============================] - 0s 2ms/step - loss: 0.2473
Epoch 92/100
32/32 [==============================] - 0s 2ms/step - loss: 0.2194
Epoch 93/100
32/32 [==============================] - 0s 2ms/step - loss: 0.2738
Epoch 94/100
32/32 [==============================] - 0s 2ms/step - loss: 0.2543
Epoch 95/100
32/32 [==============================] - 0s 2ms/step - loss: 0.2140
Epoch 96/100
32/32 [==============================] - 0s 2ms/step - loss: 0.2137
Epoch 97/100
32/32 [==============================] - 0s 2ms/step - loss: 0.2070
Epoch 98/100
32/32 [==============================] - 0s 2ms/step - loss: 0.3414
Epoch 99/100
32/32 [==============================] - 0s 2ms/step - loss: 0.1702
Epoch 100/100
32/32 [==============================] - 0s 2ms/step - loss: 0.1834
```

```
# Test the model
x_test = np.linspace(x_min, x_max, num=1000)
y_true = my_func(x_test)
y_pred = np.zeros((len(x_test), n_terms))

for i in range(n_terms):
    y_pred[:, i] = ((-1)**i / np.math.factorial(2*i + 1)) * np.power(x_test, 2*i + 1)

y_pred_transformed = model.predict(x_test)

print("True Function Values:")
print(y_true)
print("Predicted Function Values:")
print(y_pred_transformed)
print("Predicted Taylor Expansion Coefficients:")
print(model.get_weights()[-1])
```

```
32/32 [==============================] - 0s 1ms/step
True Function Values:
[ 0.00000000e+00  6.28943332e-03  1.25786178e-02  1.88673048e-02
  2.51552454e-02  3.14421909e-02  3.77278927e-02  4.40121020e-02
  5.02945704e-02  5.65750492e-02  6.28532900e-02  6.91290446e-02
  7.54020646e-02  8.16721019e-02  8.79389084e-02  9.42022363e-02
  1.00461838e-01  1.06717465e-01  1.12968871e-01  1.19215809e-01
  1.25458030e-01  1.31695289e-01  1.37927338e-01  1.44153931e-01
  1.50374822e-01  1.56589764e-01  1.62798512e-01  1.69000820e-01
  1.75196443e-01  1.81385136e-01  1.87566653e-01  1.93740751e-01
  1.99907185e-01  2.06065711e-01  2.12216086e-01  2.18358066e-01
  2.24491409e-01  2.30615871e-01  2.36731210e-01  2.42837185e-01
  2.48933554e-01  2.55020076e-01  2.61096510e-01  2.67162616e-01
  2.73218154e-01  2.79262883e-01  2.85296566e-01  2.91318963e-01
  2.97329837e-01  3.03328948e-01  3.09316061e-01  3.15290939e-01
  3.21253344e-01  3.27203041e-01  3.33139795e-01  3.39063370e-01
  3.44973534e-01  3.50870051e-01  3.56752688e-01  3.62621214e-01
  3.68475395e-01  3.74315000e-01  3.80139798e-01  3.85949559e-01
  3.91744053e-01  3.97523050e-01  4.03286322e-01  4.09033642e-01
  4.14764781e-01  4.20479513e-01  4.26177612e-01  4.31858853e-01
  4.37523010e-01  4.43169861e-01  4.48799180e-01  4.54410746e-01
  4.60004337e-01  4.65579732e-01  4.71136709e-01  4.76675049e-01
  4.82194534e-01  4.87694944e-01  4.93176062e-01  4.98637671e-01
  5.04079556e-01  5.09501500e-01  5.14903290e-01  5.20284712e-01
  5.25645553e-01  5.30985600e-01  5.36304643e-01  5.41602472e-01
  5.46878875e-01  5.52133646e-01  5.57366576e-01  5.62577458e-01
  5.67766086e-01  5.72932255e-01  5.78075760e-01  5.83196397e-01
  5.88293965e-01  5.93368262e-01  5.98419086e-01  6.03446239e-01
  6.08449521e-01  6.13428734e-01  6.18383682e-01  6.23314168e-01
  6.28219997e-01  6.33100976e-01  6.37956911e-01  6.42787610e-01
  6.47592882e-01  6.52372537e-01  6.57126385e-01  6.61854240e-01
  6.66555913e-01  6.71231219e-01  6.75879973e-01  6.80501991e-01
  6.85097090e-01  6.89665089e-01  6.94205806e-01  6.98719062e-01
  7.03204679e-01  7.07662479e-01  7.12092285e-01  7.16493923e-01
  7.20867219e-01  7.25211999e-01  7.29528091e-01  7.33815325e-01
  7.38073532e-01  7.42302542e-01  7.46502188e-01  7.50672305e-01
  7.54812728e-01  7.58923291e-01  7.63003834e-01  7.67054195e-01
  7.71074213e-01  7.75063729e-01  7.79022586e-01  7.82950626e-01
  7.86847695e-01  7.90713639e-01  7.94548304e-01  7.98351539e-01
  8.02123193e-01  8.05863117e-01  8.09571163e-01  8.13247185e-01
  8.16891037e-01  8.20502575e-01  8.24081656e-01  8.27628139e-01
  8.31141882e-01  8.34622748e-01  8.38070599e-01  8.41485297e-01
  8.44866709e-01  8.48214700e-01  8.51529138e-01  8.54809891e-01
  8.58056831e-01  8.61269828e-01  8.64448755e-01  8.67593488e-01
  8.70703900e-01  8.73779870e-01  8.76821275e-01  8.79827996e-01
  8.82799913e-01  8.85736908e-01  8.88638867e-01  8.91505673e-01
  8.94337213e-01  8.97133376e-01  8.99894051e-01  9.02619128e-01
  9.05308500e-01  9.07962060e-01  9.10579704e-01  9.13161327e-01
  9.15706829e-01  9.18216107e-01  9.20689063e-01  9.23125599e-01
  9.25525619e-01  9.27889027e-01  9.30215731e-01  9.32505637e-01
  9.34758657e-01  9.36974699e-01  9.39153678e-01  9.41295506e-01
```

```
9.43400098e-01 9.45467373e-01 9.47497247e-01 9.49489640e-01
9.51444475e-01 9.53361672e-01 9.55241158e-01 9.57082856e-01
9.58886695e-01 9.60652602e-01 9.62380509e-01 9.64070347e-01
9.65722048e-01 9.67335548e-01 9.68910783e-01 9.70447691e-01
9.71946209e-01 9.73406281e-01 9.74827847e-01 9.76210851e-01
9.77555239e-01 9.78860957e-01 9.80127955e-01 9.81356181e-01
9.82545587e-01 9.83696126e-01 9.84807753e-01 9.85880423e-01
```