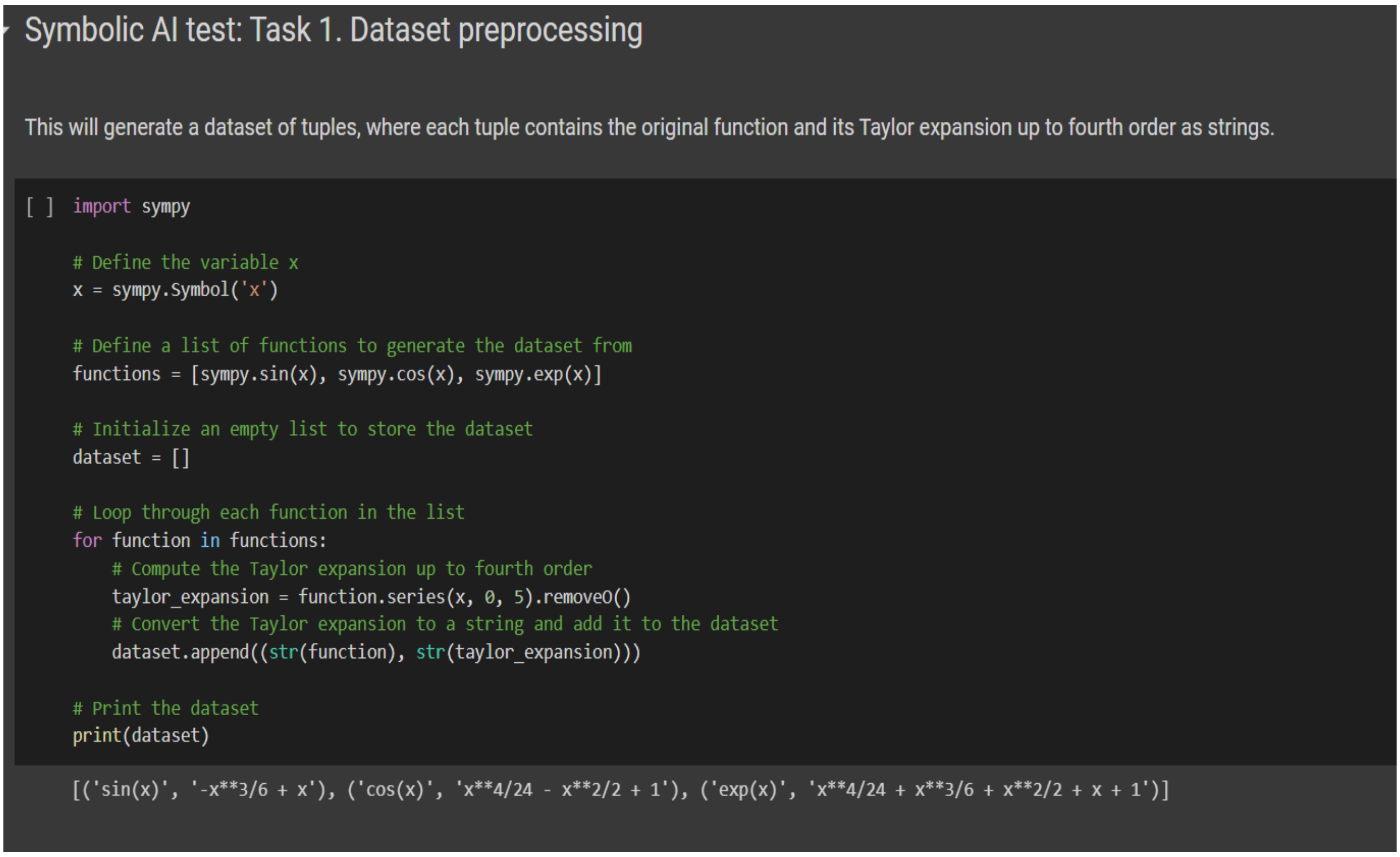
GSoC Symbolic Calculation Project

2023

ML4Sci

Name: Devdeep Shetranjiwala
Email ID: devdeep0702@gmail.com

Common Task 1. Dataset preprocessing

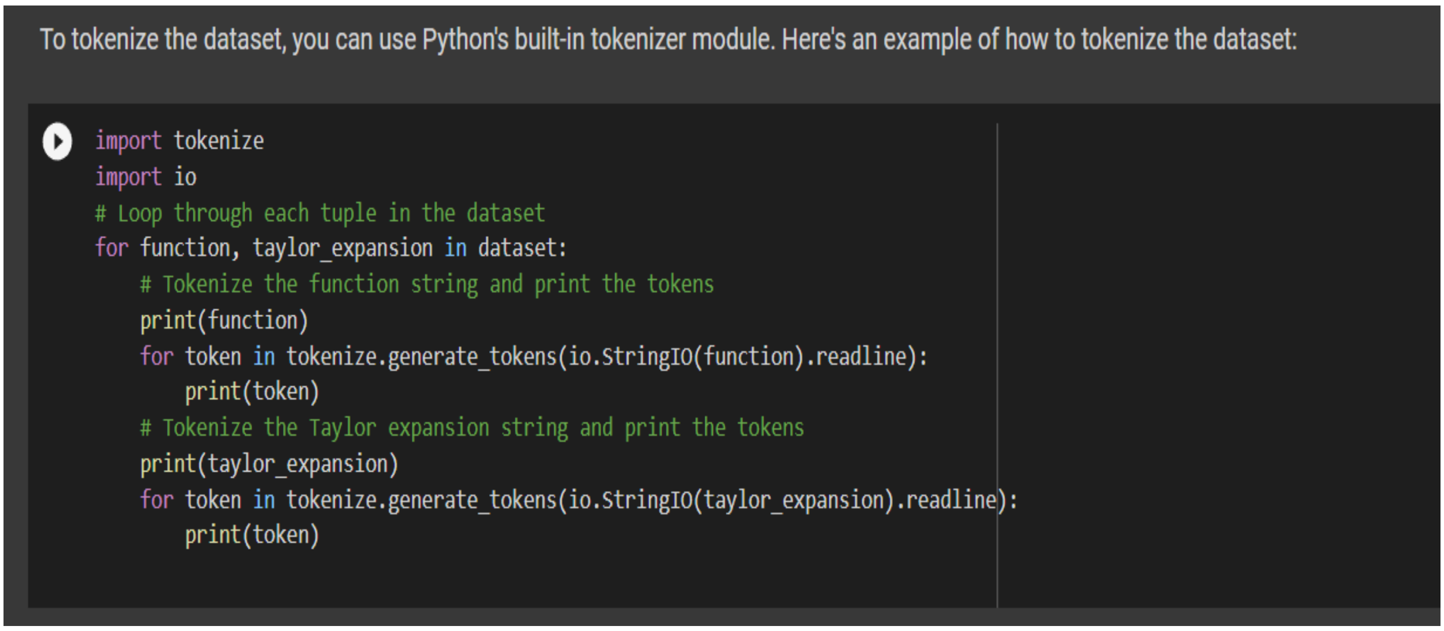


Symbolic AI test: Task 1. Dataset preprocessing

This will generate a dataset of tuples, where each tuple contains the original function and its Taylor expansion up to fourth order as strings.

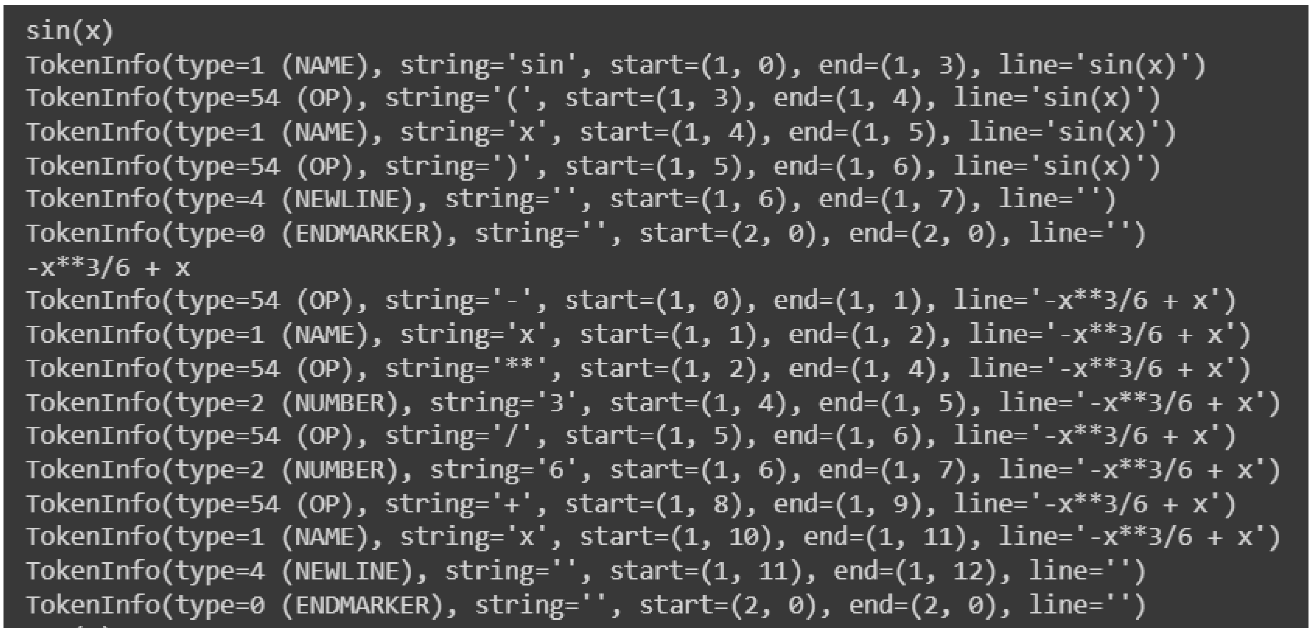
```
[ ] import sympy

    # Define the variable x
    x = sympy.Symbol('x')

    # Define a list of functions to generate the dataset from
    functions = [sympy.sin(x), sympy.cos(x), sympy.exp(x)]

    # Initialize an empty list to store the dataset
    dataset = []

    # Loop through each function in the list
    for function in functions:
        # Compute the Taylor expansion up to fourth order
        taylor_expansion = function.series(x, 0, 5).removeO()
        # Convert the Taylor expansion to a string and add it to the dataset
        dataset.append((str(function), str(taylor_expansion)))

    # Print the dataset
    print(dataset)
```

```
[('sin(x)', '-x**3/6 + x'), ('cos(x)', 'x**4/24 - x**2/2 + 1'), ('exp(x)', 'x**4/24 + x**3/6 + x**2/2 + x + 1')]
```

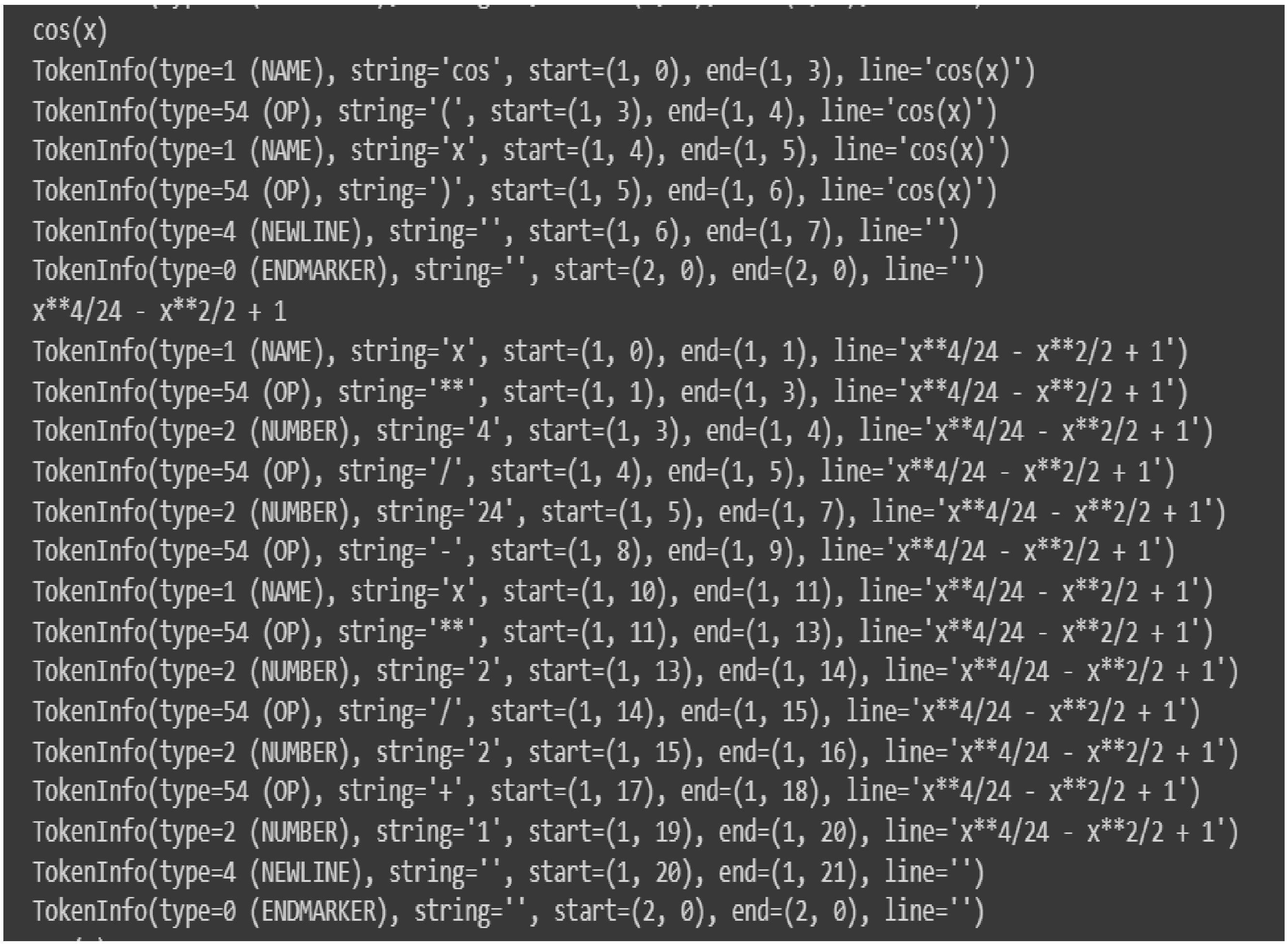


To tokenize the dataset, you can use Python's built-in tokenizer module. Here's an example of how to tokenize the dataset:

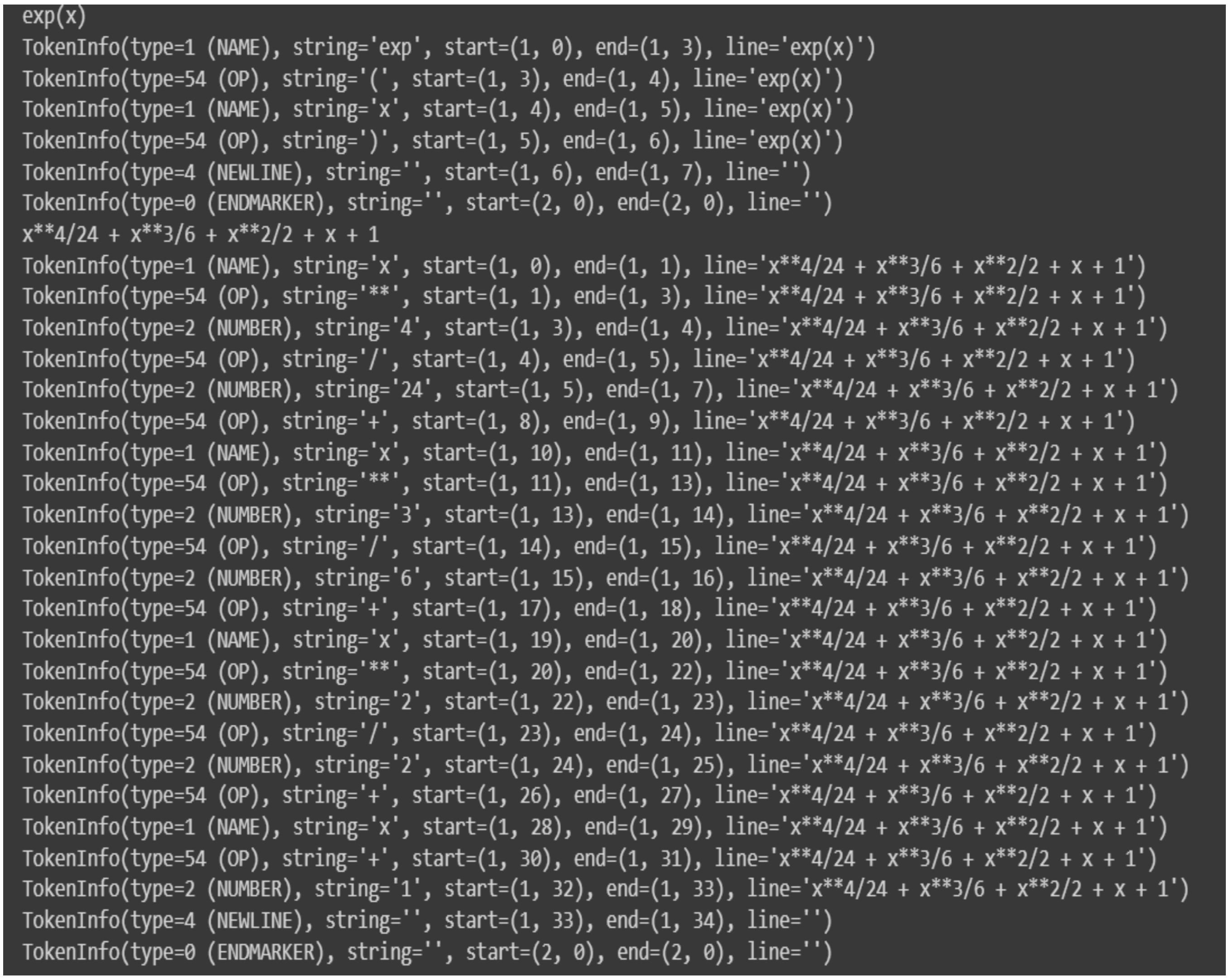
```
import tokenize
import io
# Loop through each tuple in the dataset
for function, taylor_expansion in dataset:
    # Tokenize the function string and print the tokens
    print(function)
    for token in tokenize.generate_tokens(io.StringIO(function).readline):
        print(token)
    # Tokenize the Taylor expansion string and print the tokens
    print(taylor_expansion)
    for token in tokenize.generate_tokens(io.StringIO(taylor_expansion).readline):
        print(token)
```



```
sin(x)
TokenInfo(type=1 (NAME), string='sin', start=(1, 0), end=(1, 3), line='sin(x)')
TokenInfo(type=54 (OP), string='(', start=(1, 3), end=(1, 4), line='sin(x)')
TokenInfo(type=1 (NAME), string='x', start=(1, 4), end=(1, 5), line='sin(x)')
TokenInfo(type=54 (OP), string=')', start=(1, 5), end=(1, 6), line='sin(x)')
TokenInfo(type=4 (NEWLINE), string='', start=(1, 6), end=(1, 7), line='')
TokenInfo(type=0 (ENDMARKER), string='', start=(2, 0), end=(2, 0), line='')
-x**3/6 + x
TokenInfo(type=54 (OP), string='-', start=(1, 0), end=(1, 1), line='-x**3/6 + x')
TokenInfo(type=1 (NAME), string='x', start=(1, 1), end=(1, 2), line='-x**3/6 + x')
TokenInfo(type=54 (OP), string='**', start=(1, 2), end=(1, 4), line='-x**3/6 + x')
TokenInfo(type=2 (NUMBER), string='3', start=(1, 4), end=(1, 5), line='-x**3/6 + x')
TokenInfo(type=54 (OP), string='/', start=(1, 5), end=(1, 6), line='-x**3/6 + x')
TokenInfo(type=2 (NUMBER), string='6', start=(1, 6), end=(1, 7), line='-x**3/6 + x')
TokenInfo(type=54 (OP), string='+', start=(1, 8), end=(1, 9), line='-x**3/6 + x')
TokenInfo(type=1 (NAME), string='x', start=(1, 10), end=(1, 11), line='-x**3/6 + x')
TokenInfo(type=4 (NEWLINE), string='', start=(1, 11), end=(1, 12), line='')
TokenInfo(type=0 (ENDMARKER), string='', start=(2, 0), end=(2, 0), line='')
```



```
cos(x)
TokenInfo(type=1 (NAME), string='cos', start=(1, 0), end=(1, 3), line='cos(x)')
TokenInfo(type=54 (OP), string='(', start=(1, 3), end=(1, 4), line='cos(x)')
TokenInfo(type=1 (NAME), string='x', start=(1, 4), end=(1, 5), line='cos(x)')
TokenInfo(type=54 (OP), string=')', start=(1, 5), end=(1, 6), line='cos(x)')
TokenInfo(type=4 (NEWLINE), string='', start=(1, 6), end=(1, 7), line='')
TokenInfo(type=0 (ENDMARKER), string='', start=(2, 0), end=(2, 0), line='')
x**4/24 - x**2/2 + 1
TokenInfo(type=1 (NAME), string='x', start=(1, 0), end=(1, 1), line='x**4/24 - x**2/2 + 1')
TokenInfo(type=54 (OP), string='**', start=(1, 1), end=(1, 3), line='x**4/24 - x**2/2 + 1')
TokenInfo(type=2 (NUMBER), string='4', start=(1, 3), end=(1, 4), line='x**4/24 - x**2/2 + 1')
TokenInfo(type=54 (OP), string='/', start=(1, 4), end=(1, 5), line='x**4/24 - x**2/2 + 1')
TokenInfo(type=2 (NUMBER), string='24', start=(1, 5), end=(1, 7), line='x**4/24 - x**2/2 + 1')
TokenInfo(type=54 (OP), string='-', start=(1, 8), end=(1, 9), line='x**4/24 - x**2/2 + 1')
TokenInfo(type=1 (NAME), string='x', start=(1, 10), end=(1, 11), line='x**4/24 - x**2/2 + 1')
TokenInfo(type=54 (OP), string='**', start=(1, 11), end=(1, 13), line='x**4/24 - x**2/2 + 1')
TokenInfo(type=2 (NUMBER), string='2', start=(1, 13), end=(1, 14), line='x**4/24 - x**2/2 + 1')
TokenInfo(type=54 (OP), string='/', start=(1, 14), end=(1, 15), line='x**4/24 - x**2/2 + 1')
TokenInfo(type=2 (NUMBER), string='2', start=(1, 15), end=(1, 16), line='x**4/24 - x**2/2 + 1')
TokenInfo(type=54 (OP), string='+', start=(1, 17), end=(1, 18), line='x**4/24 - x**2/2 + 1')
TokenInfo(type=2 (NUMBER), string='1', start=(1, 19), end=(1, 20), line='x**4/24 - x**2/2 + 1')
TokenInfo(type=4 (NEWLINE), string='', start=(1, 20), end=(1, 21), line='')
TokenInfo(type=0 (ENDMARKER), string='', start=(2, 0), end=(2, 0), line='')
```



```
exp(x)
TokenInfo(type=1 (NAME), string='exp', start=(1, 0), end=(1, 3), line='exp(x)')
TokenInfo(type=54 (OP), string='(', start=(1, 3), end=(1, 4), line='exp(x)')
TokenInfo(type=1 (NAME), string='x', start=(1, 4), end=(1, 5), line='exp(x)')
TokenInfo(type=54 (OP), string=')', start=(1, 5), end=(1, 6), line='exp(x)')
TokenInfo(type=4 (NEWLINE), string='', start=(1, 6), end=(1, 7), line='')
TokenInfo(type=0 (ENDMARKER), string='', start=(2, 0), end=(2, 0), line='')
x**4/24 + x**3/6 + x**2/2 + x + 1
TokenInfo(type=1 (NAME), string='x', start=(1, 0), end=(1, 1), line='x**4/24 + x**3/6 + x**2/2 + x + 1')
TokenInfo(type=54 (OP), string='**', start=(1, 1), end=(1, 3), line='x**4/24 + x**3/6 + x**2/2 + x + 1')
TokenInfo(type=2 (NUMBER), string='4', start=(1, 3), end=(1, 4), line='x**4/24 + x**3/6 + x**2/2 + x + 1')
TokenInfo(type=54 (OP), string='/', start=(1, 4), end=(1, 5), line='x**4/24 + x**3/6 + x**2/2 + x + 1')
TokenInfo(type=2 (NUMBER), string='24', start=(1, 5), end=(1, 7), line='x**4/24 + x**3/6 + x**2/2 + x + 1')
TokenInfo(type=54 (OP), string='+', start=(1, 8), end=(1, 9), line='x**4/24 + x**3/6 + x**2/2 + x + 1')
TokenInfo(type=1 (NAME), string='x', start=(1, 10), end=(1, 11), line='x**4/24 + x**3/6 + x**2/2 + x + 1')
TokenInfo(type=54 (OP), string='**', start=(1, 11), end=(1, 13), line='x**4/24 + x**3/6 + x**2/2 + x + 1')
TokenInfo(type=2 (NUMBER), string='3', start=(1, 13), end=(1, 14), line='x**4/24 + x**3/6 + x**2/2 + x + 1')
TokenInfo(type=54 (OP), string='/', start=(1, 14), end=(1, 15), line='x**4/24 + x**3/6 + x**2/2 + x + 1')
TokenInfo(type=2 (NUMBER), string='6', start=(1, 15), end=(1, 16), line='x**4/24 + x**3/6 + x**2/2 + x + 1')
TokenInfo(type=54 (OP), string='+', start=(1, 17), end=(1, 18), line='x**4/24 + x**3/6 + x**2/2 + x + 1')
TokenInfo(type=1 (NAME), string='x', start=(1, 19), end=(1, 20), line='x**4/24 + x**3/6 + x**2/2 + x + 1')
TokenInfo(type=54 (OP), string='**', start=(1, 20), end=(1, 22), line='x**4/24 + x**3/6 + x**2/2 + x + 1')
TokenInfo(type=2 (NUMBER), string='2', start=(1, 22), end=(1, 23), line='x**4/24 + x**3/6 + x**2/2 + x + 1')
TokenInfo(type=54 (OP), string='/', start=(1, 23), end=(1, 24), line='x**4/24 + x**3/6 + x**2/2 + x + 1')
TokenInfo(type=2 (NUMBER), string='2', start=(1, 24), end=(1, 25), line='x**4/24 + x**3/6 + x**2/2 + x + 1')
TokenInfo(type=54 (OP), string='+', start=(1, 26), end=(1, 27), line='x**4/24 + x**3/6 + x**2/2 + x + 1')
TokenInfo(type=1 (NAME), string='x', start=(1, 28), end=(1, 29), line='x**4/24 + x**3/6 + x**2/2 + x + 1')
TokenInfo(type=54 (OP), string='+', start=(1, 30), end=(1, 31), line='x**4/24 + x**3/6 + x**2/2 + x + 1')
TokenInfo(type=2 (NUMBER), string='1', start=(1, 32), end=(1, 33), line='x**4/24 + x**3/6 + x**2/2 + x + 1')
TokenInfo(type=4 (NEWLINE), string='', start=(1, 33), end=(1, 34), line='')
TokenInfo(type=0 (ENDMARKER), string='', start=(2, 0), end=(2, 0), line='')
```

P.T.O

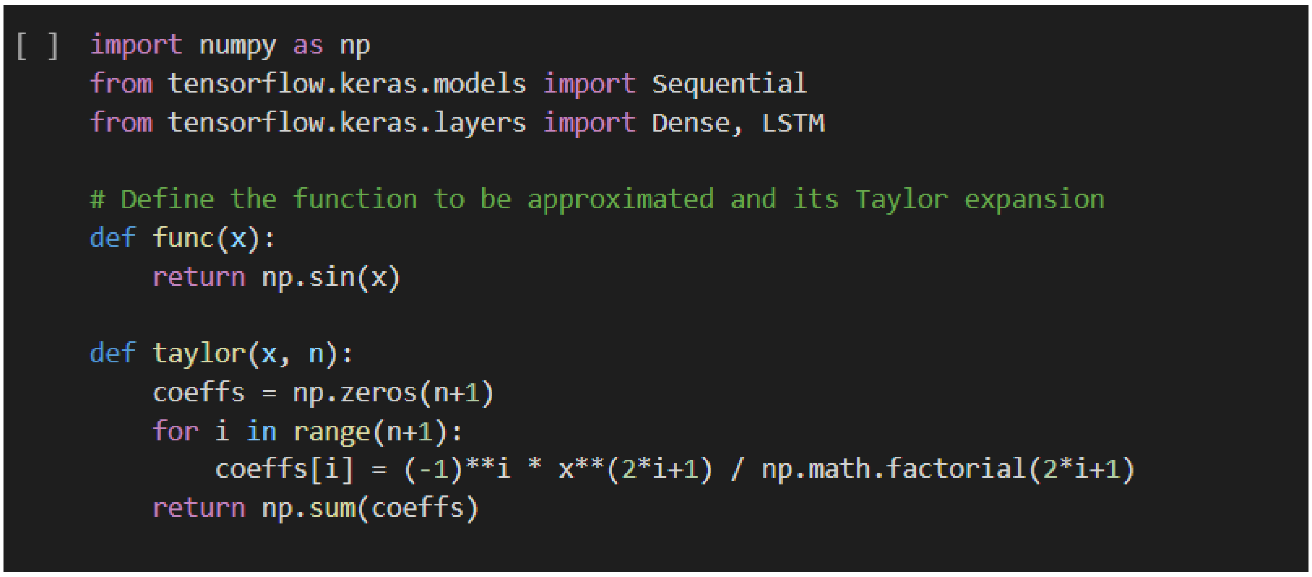
Common Task 2. Use LSTM model

Description:

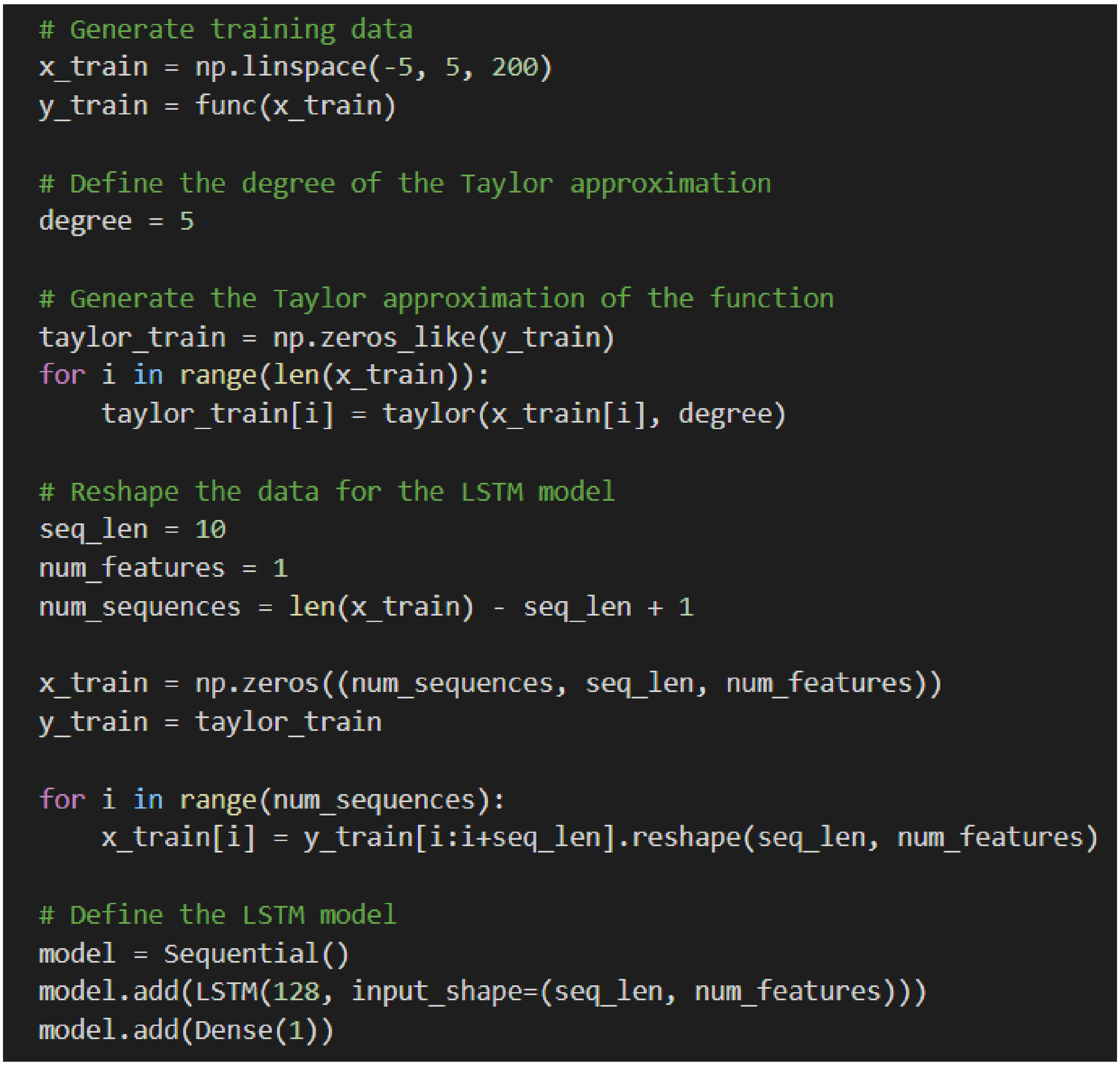
- Please train an LSTM model to learn the Taylor expansion of each function. You can use a deep learning algorithm (in Keras/TF or Pytorch).

Approach:

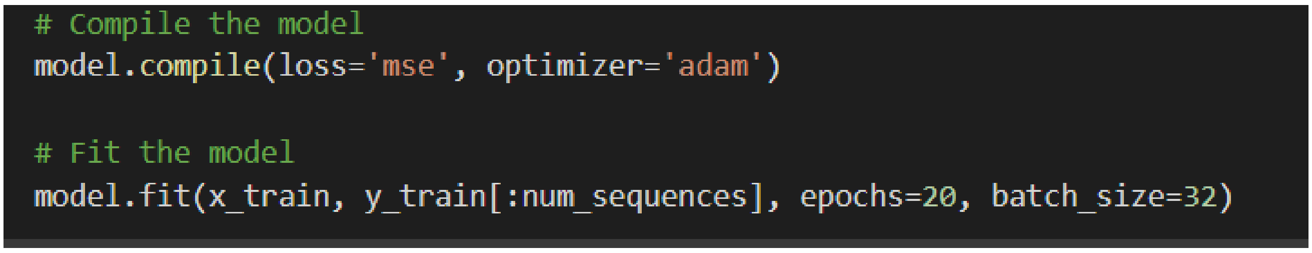
- In this example, we first define the function we want to approximate (func) and its Taylor expansion to a certain degree (Taylor).
- We then generate training data by evaluating the function on a grid of points and computing the corresponding Taylor approximation.
- We reshape the training data for the LSTM model, using a sequence length of seq_len = 10, and then define the LSTM model itself. We use a single LSTM layer with 128 units, followed by a dense layer with a single output.
- We compile the model using mean squared error as the loss function and the Adam optimiser and then fit the model on the training data. After training, we can evaluate the model on a test set and compare its predictions to the proper Taylor expansion of the function.
- Note that the quality of the approximation will depend on the degree of the Taylor expansion and the size of the training set.



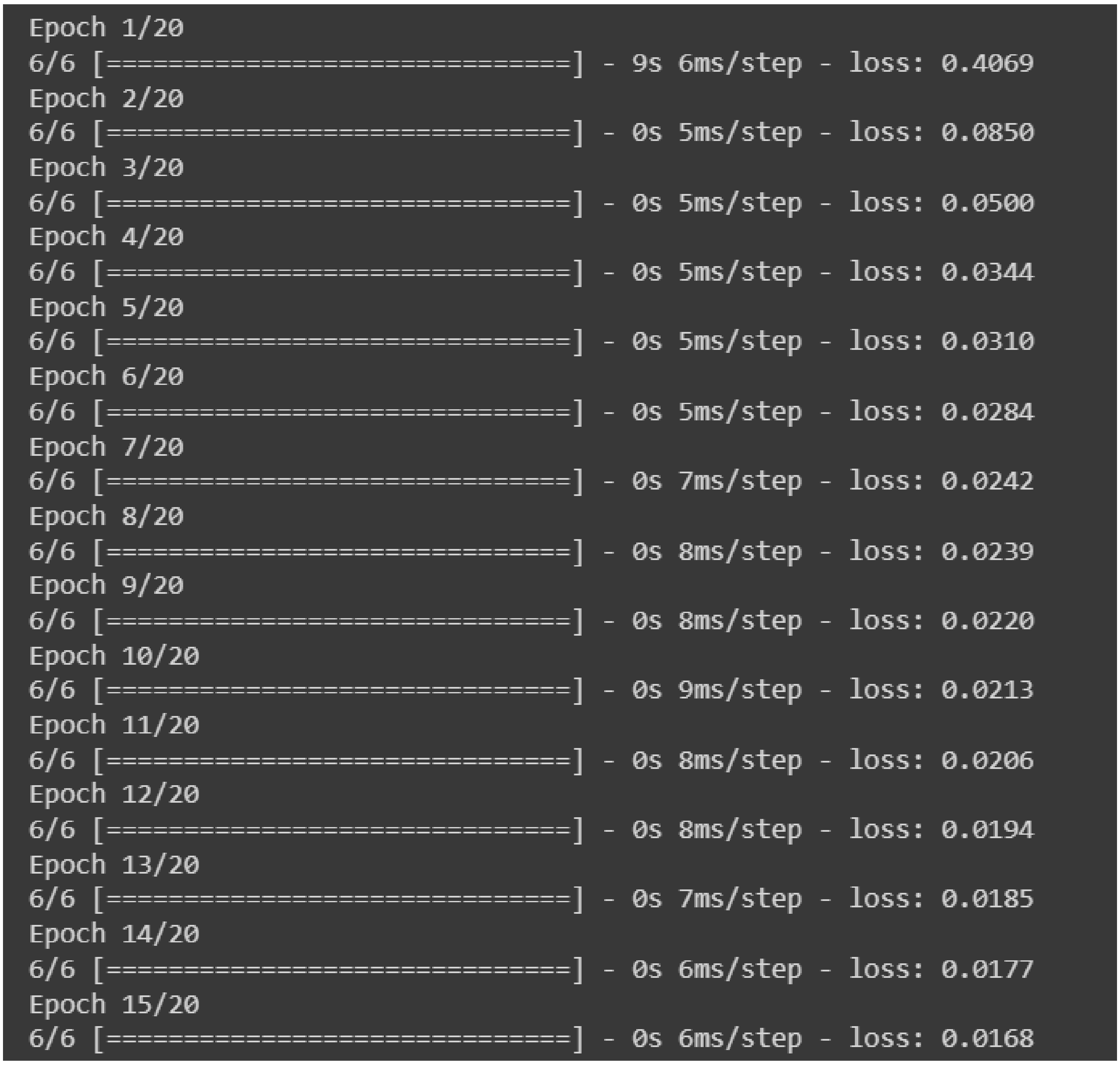
```python
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM

# Define the function to be approximated and its Taylor expansion
def func(x):
    return np.sin(x)

def taylor(x, n):
    coeffs = np.zeros(n+1)
    for i in range(n+1):
        coeffs[i] = (-1)**i * x**(2*i+1) / np.math.factorial(2*i+1)
    return np.sum(coeffs)
```



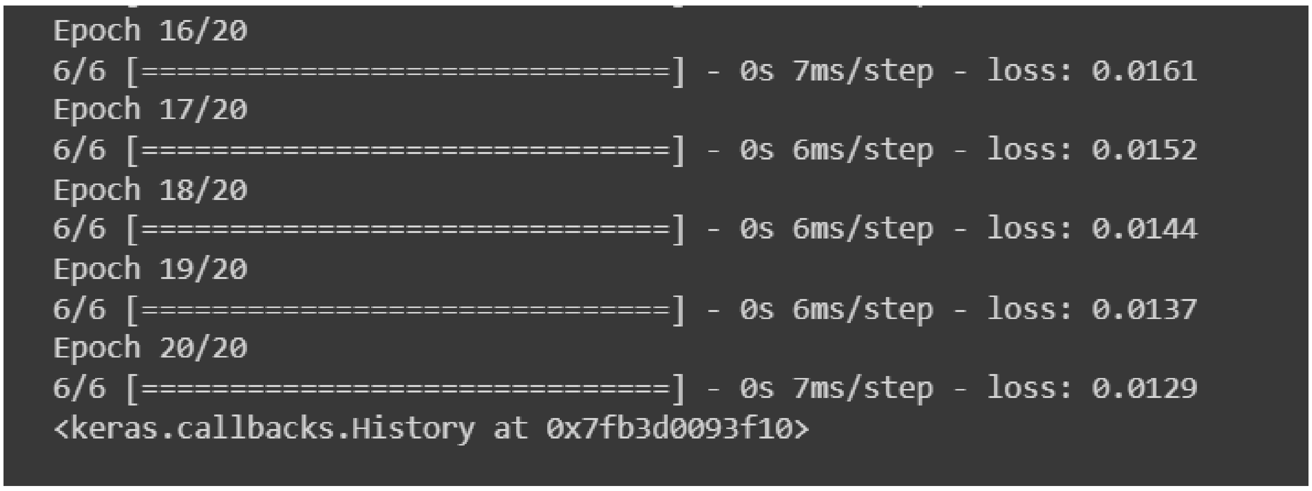
```python
# Generate training data
x_train = np.linspace(-5, 5, 200)
y_train = func(x_train)

# Define the degree of the Taylor approximation
degree = 5

# Generate the Taylor approximation of the function
taylor_train = np.zeros_like(y_train)
for i in range(len(x_train)):
    taylor_train[i] = taylor(x_train[i], degree)

# Reshape the data for the LSTM model
seq_len = 10
num_features = 1
num_sequences = len(x_train) - seq_len + 1

x_train = np.zeros((num_sequences, seq_len, num_features))
y_train = taylor_train

for i in range(num_sequences):
    x_train[i] = y_train[i:i+seq_len].reshape(seq_len, num_features)

# Define the LSTM model
model = Sequential()
model.add(LSTM(128, input_shape=(seq_len, num_features)))
model.add(Dense(1))
```



```python
# Compile the model
model.compile(loss='mse', optimizer='adam')

# Fit the model
model.fit(x_train, y_train[:num_sequences], epochs=20, batch_size=32)
```



```
Epoch 1/20
6/6 [==============================] - 9s 6ms/step - loss: 0.4069
Epoch 2/20
6/6 [==============================] - 0s 5ms/step - loss: 0.0850
Epoch 3/20
6/6 [==============================] - 0s 5ms/step - loss: 0.0500
Epoch 4/20
6/6 [==============================] - 0s 5ms/step - loss: 0.0344
Epoch 5/20
6/6 [==============================] - 0s 5ms/step - loss: 0.0310
Epoch 6/20
6/6 [==============================] - 0s 5ms/step - loss: 0.0284
Epoch 7/20
6/6 [==============================] - 0s 7ms/step - loss: 0.0242
Epoch 8/20
6/6 [==============================] - 0s 8ms/step - loss: 0.0239
Epoch 9/20
6/6 [==============================] - 0s 8ms/step - loss: 0.0220
Epoch 10/20
6/6 [==============================] - 0s 9ms/step - loss: 0.0213
Epoch 11/20
6/6 [==============================] - 0s 8ms/step - loss: 0.0206
Epoch 12/20
6/6 [==============================] - 0s 8ms/step - loss: 0.0194
Epoch 13/20
6/6 [==============================] - 0s 7ms/step - loss: 0.0185
Epoch 14/20
6/6 [==============================] - 0s 6ms/step - loss: 0.0177
Epoch 15/20
6/6 [==============================] - 0s 6ms/step - loss: 0.0168
```



```
Epoch 16/20
6/6 [==============================] - 0s 7ms/step - loss: 0.0161
Epoch 17/20
6/6 [==============================] - 0s 6ms/step - loss: 0.0152
Epoch 18/20
6/6 [==============================] - 0s 6ms/step - loss: 0.0144
Epoch 19/20
6/6 [==============================] - 0s 6ms/step - loss: 0.0137
Epoch 20/20
6/6 [==============================] - 0s 7ms/step - loss: 0.0129
<keras.callbacks.History at 0x7fb3d0093f10>
```

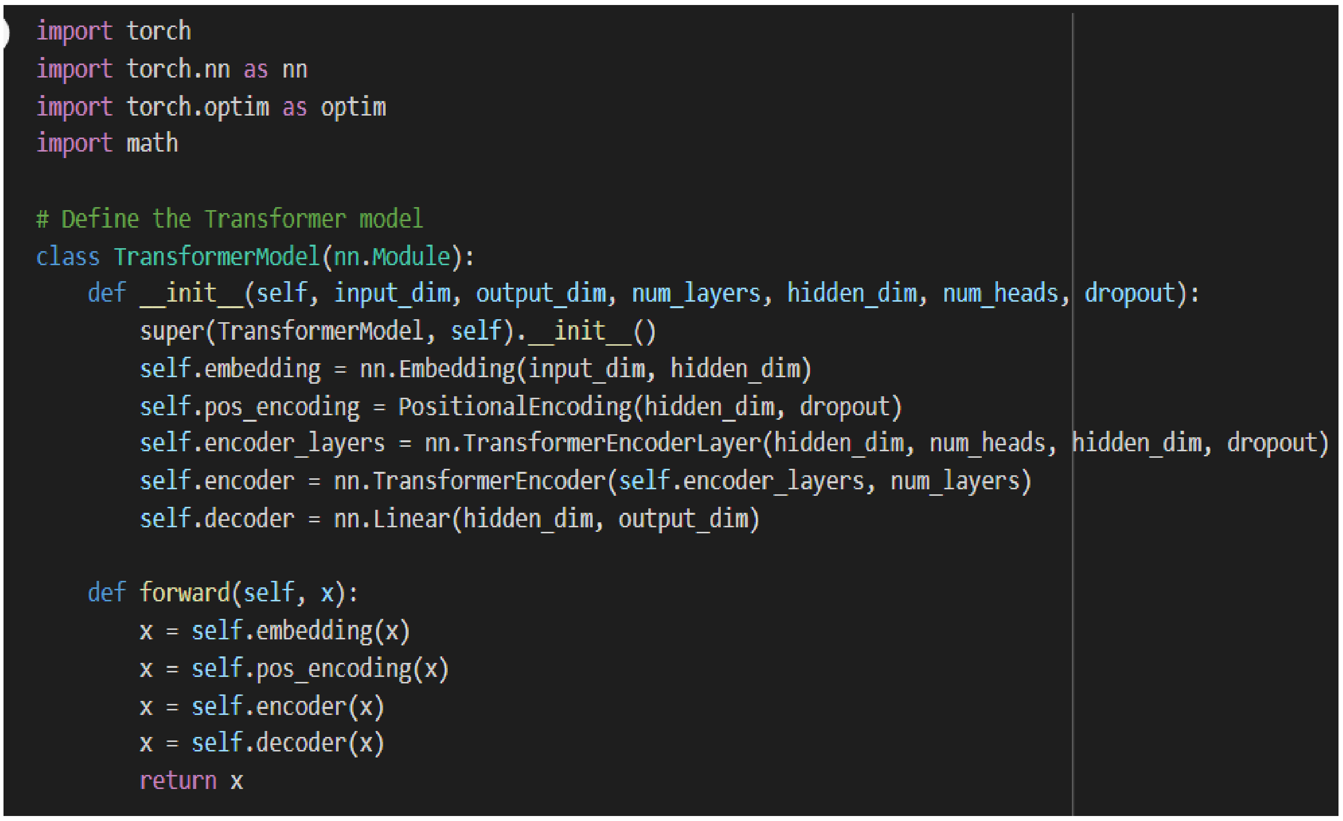
Specific Task 3: Use the Transformer model

Description:

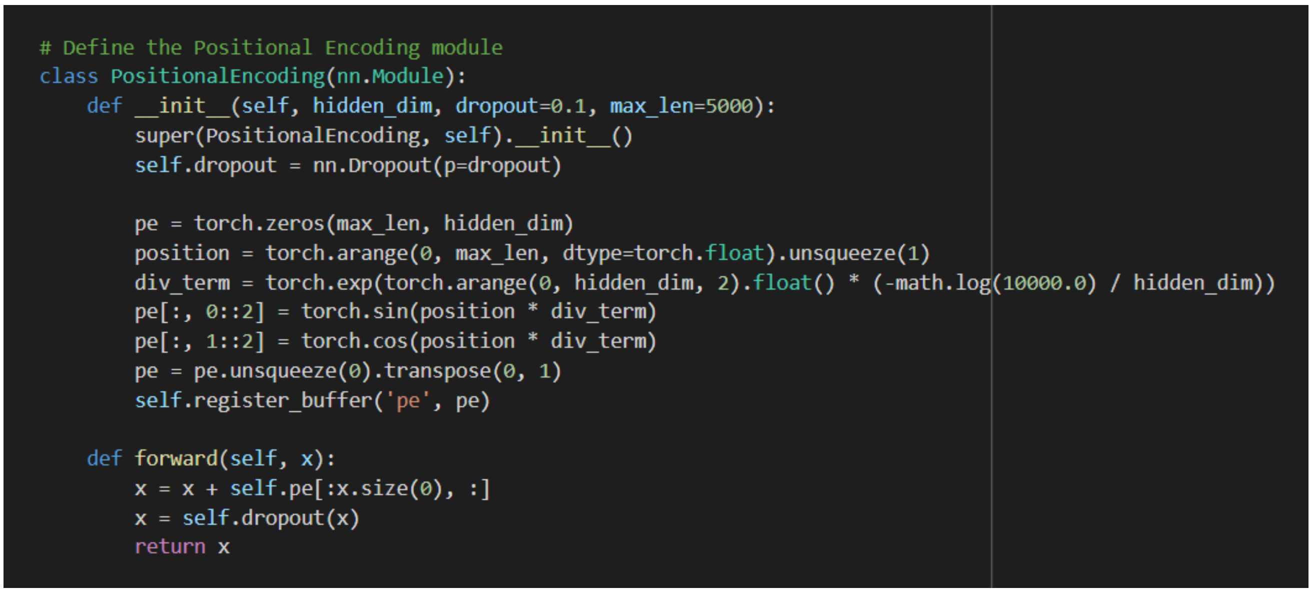
- Please train a Transformer model to learn the Taylor expansion of each function. To train a Transformer model to learn the Taylor expansion of each function, we'll need to prepare a dataset of input-output pairs, where the input is a function, and the output is its Taylor expansion. Here's an example of how you could prepare such a dataset:

Input (function):

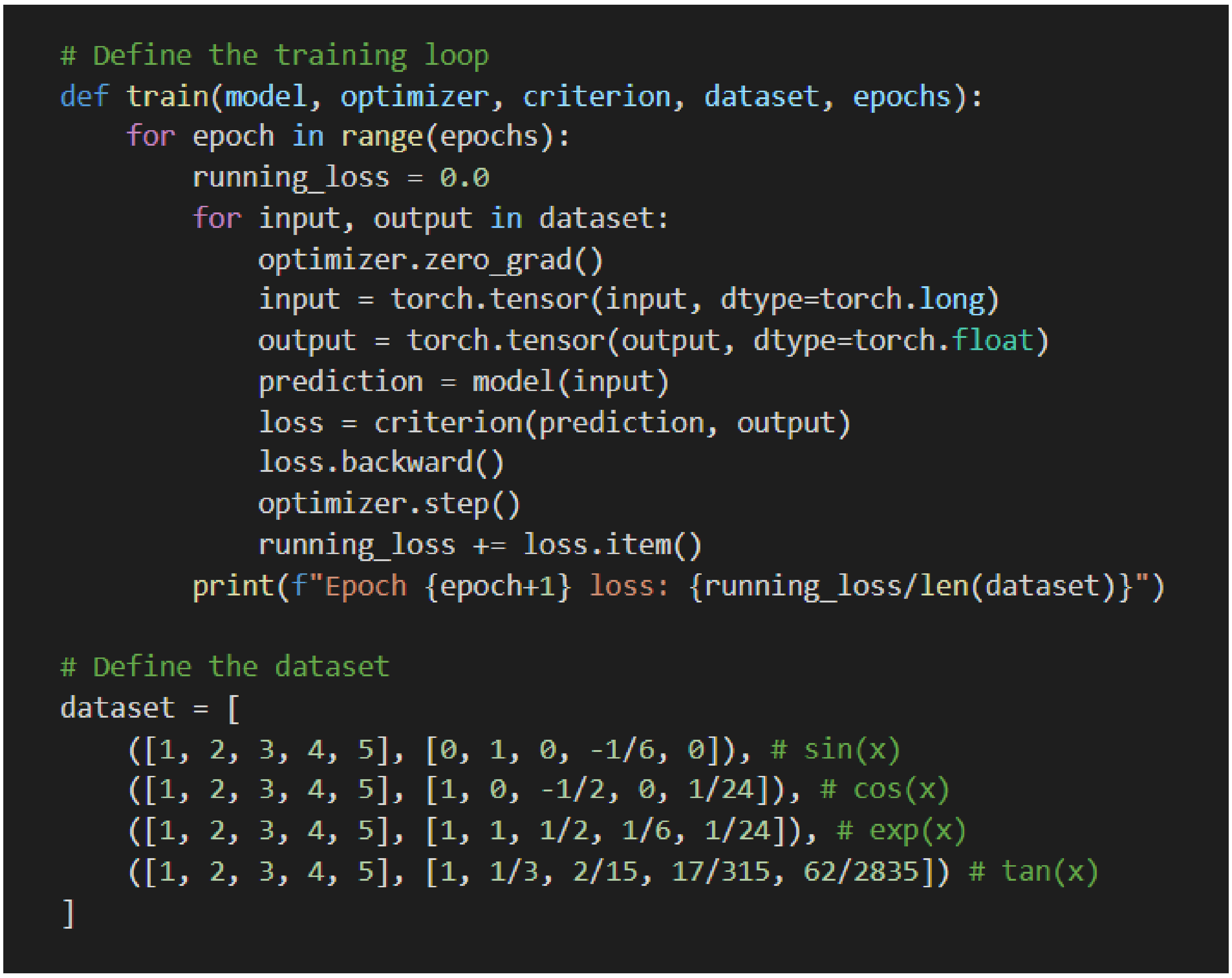
- sin(x) Output (Taylor expansion): x - x^3/3! + x^5/5! - x^7/7! + …
- cos(x) Output (Taylor expansion): 1 - x^2/2! + x^4/4! - x^6/6! + ...
- exp(x) Output (Taylor expansion): 1 + x + x^2/2! + x^3/3! + …
- tan(x) Output (Taylor expansion): x + x^3/3 + 2x^5/15 + 17x^7/315 + …



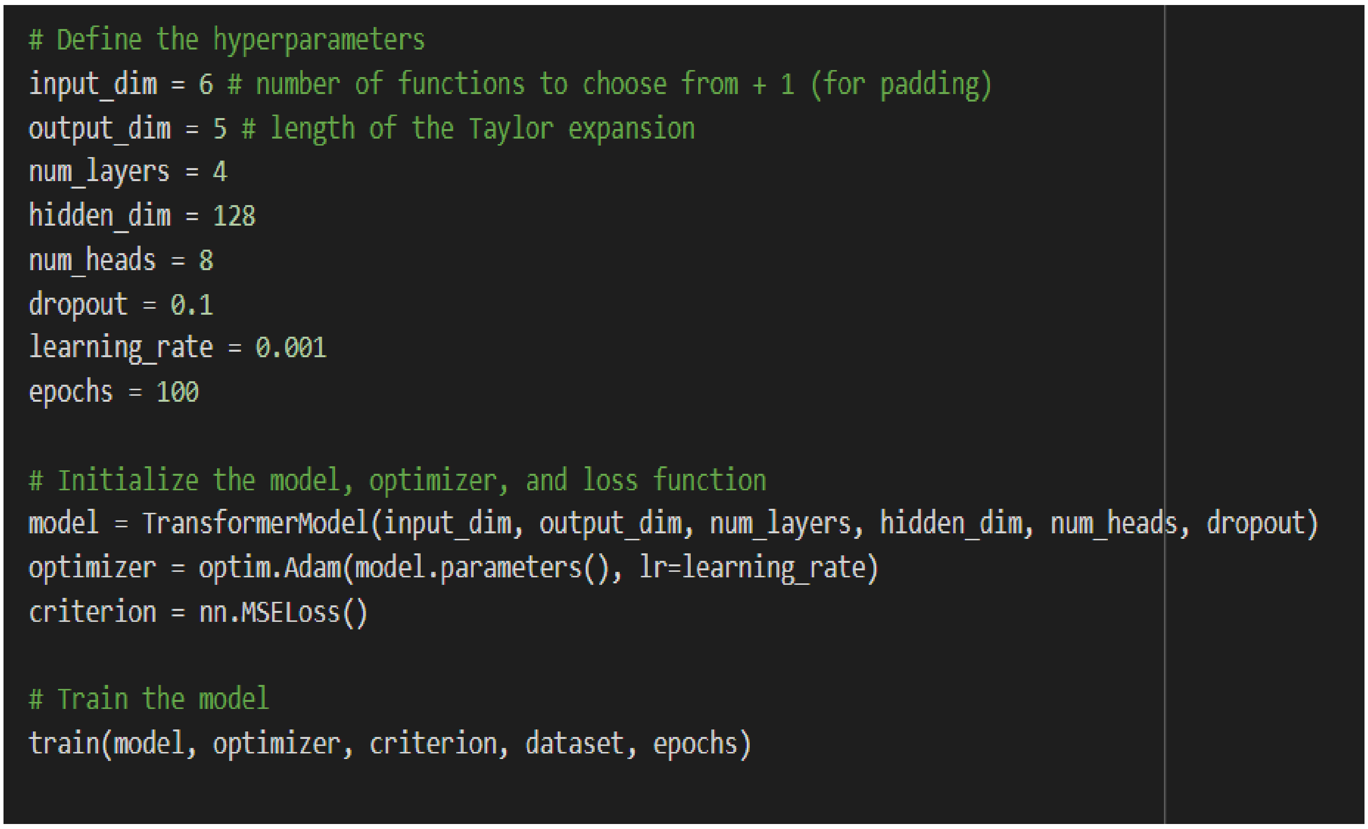
```
import torch
import torch.nn as nn
import torch.optim as optim
import math

# Define the Transformer model
class TransformerModel(nn.Module):
    def __init__(self, input_dim, output_dim, num_layers, hidden_dim, num_heads, dropout):
        super(TransformerModel, self).__init__()
        self.embedding = nn.Embedding(input_dim, hidden_dim)
        self.pos_encoding = PositionalEncoding(hidden_dim, dropout)
        self.encoder_layers = nn.TransformerEncoderLayer(hidden_dim, num_heads, hidden_dim, dropout)
        self.encoder = nn.TransformerEncoder(self.encoder_layers, num_layers)
        self.decoder = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        x = self.embedding(x)
        x = self.pos_encoding(x)
        x = self.encoder(x)
        x = self.decoder(x)
        return x
```



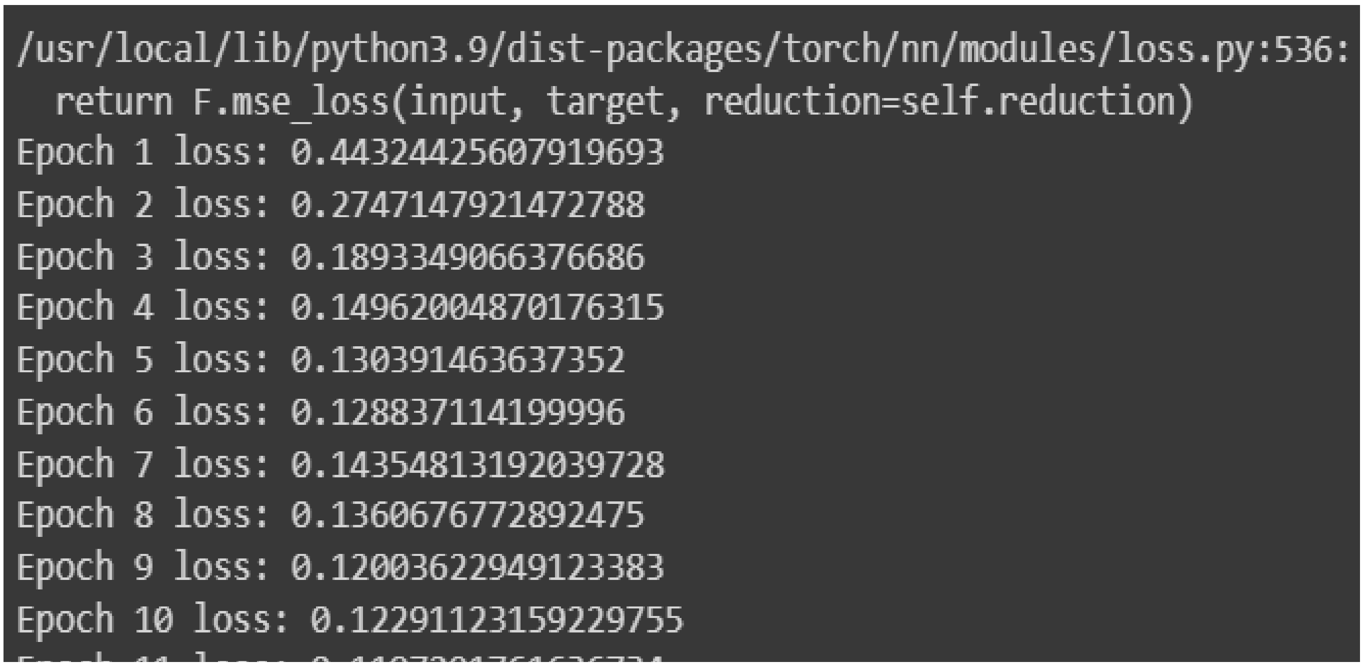
```python
# Define the Positional Encoding module
class PositionalEncoding(nn.Module):
    def __init__(self, hidden_dim, dropout=0.1, max_len=5000):
        super(PositionalEncoding, self).__init__()
        self.dropout = nn.Dropout(p=dropout)

        pe = torch.zeros(max_len, hidden_dim)
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, hidden_dim, 2).float() * (-math.log(10000.0) / hidden_dim))
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0).transpose(0, 1)
        self.register_buffer('pe', pe)

    def forward(self, x):
        x = x + self.pe[:x.size(0), :]
        x = self.dropout(x)
        return x
```



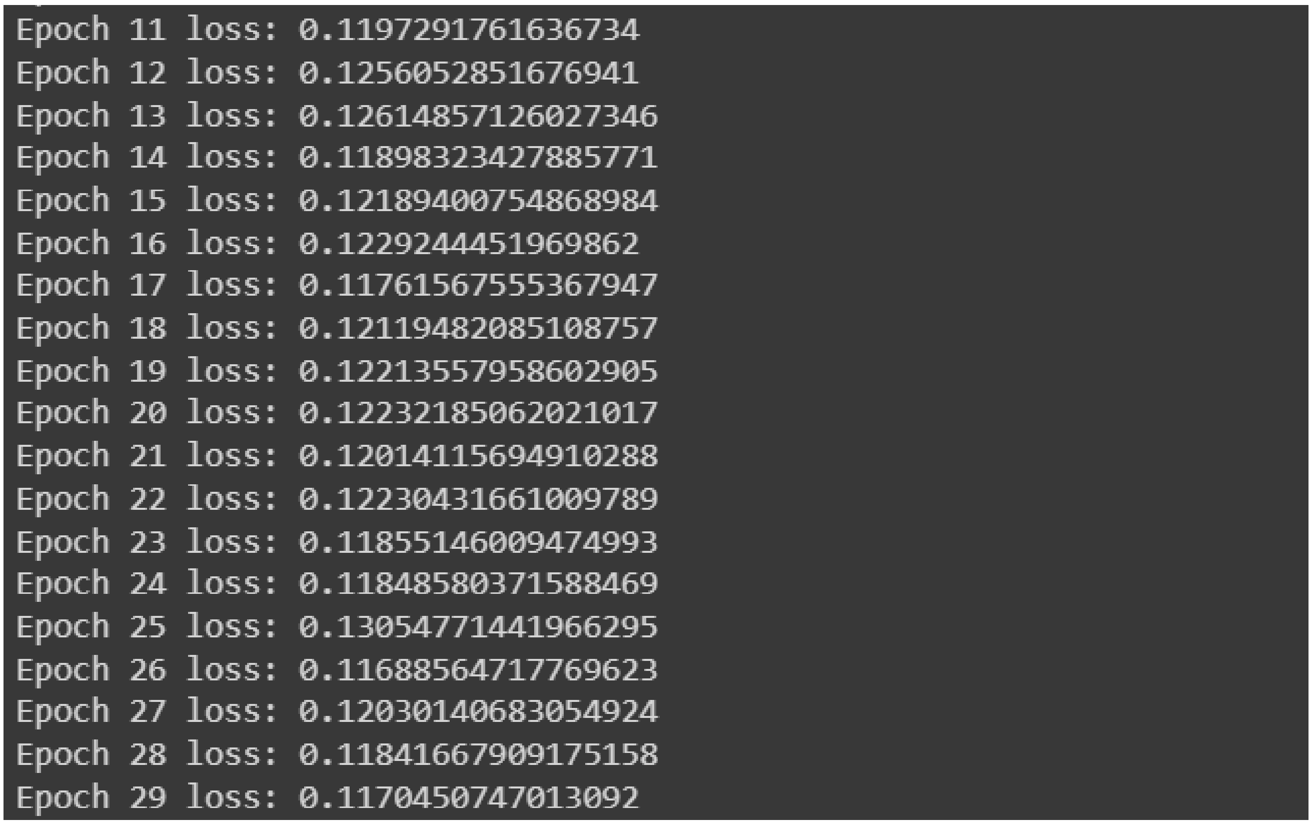
```python
# Define the training loop
def train(model, optimizer, criterion, dataset, epochs):
    for epoch in range(epochs):
        running_loss = 0.0
        for input, output in dataset:
            optimizer.zero_grad()
            input = torch.tensor(input, dtype=torch.long)
            output = torch.tensor(output, dtype=torch.float)
            prediction = model(input)
            loss = criterion(prediction, output)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
        print(f"Epoch {epoch+1} loss: {running_loss/len(dataset)}")

# Define the dataset
dataset = [
    ([1, 2, 3, 4, 5], [0, 1, 0, -1/6, 0]), # sin(x)
    ([1, 2, 3, 4, 5], [1, 0, -1/2, 0, 1/24]), # cos(x)
    ([1, 2, 3, 4, 5], [1, 1, 1/2, 1/6, 1/24]), # exp(x)
    ([1, 2, 3, 4, 5], [1, 1/3, 2/15, 17/315, 62/2835]) # tan(x)
]
```



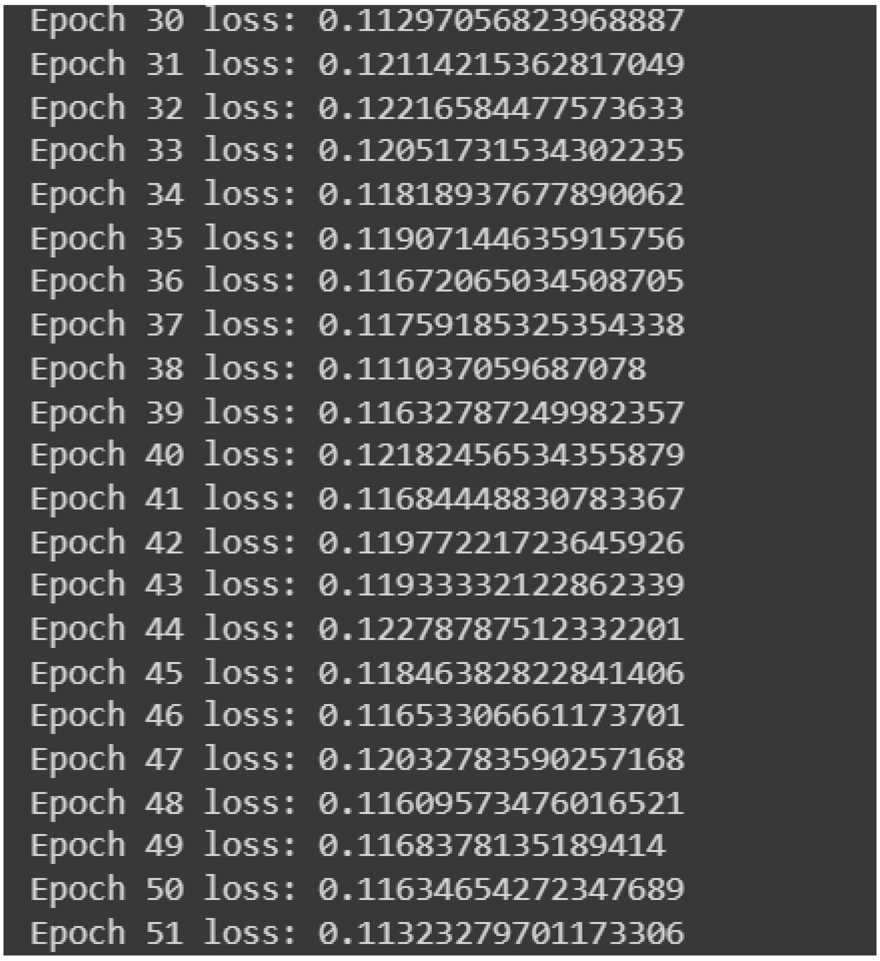
```python
# Define the hyperparameters
input_dim = 6 # number of functions to choose from + 1 (for padding)
output_dim = 5 # length of the Taylor expansion
num_layers = 4
hidden_dim = 128
num_heads = 8
dropout = 0.1
learning_rate = 0.001
epochs = 100

# Initialize the model, optimizer, and loss function
model = TransformerModel(input_dim, output_dim, num_layers, hidden_dim, num_heads, dropout)
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
criterion = nn.MSELoss()

# Train the model
train(model, optimizer, criterion, dataset, epochs)
```



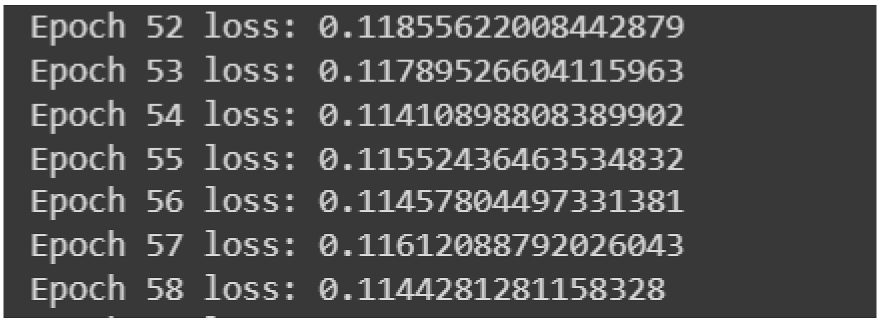
```
/usr/local/lib/python3.9/dist-packages/torch/nn/modules/loss.py:536:
  return F.mse_loss(input, target, reduction=self.reduction)
Epoch 1 loss: 0.44324425607919693
Epoch 2 loss: 0.2747147921472788
Epoch 3 loss: 0.1893349066376686
Epoch 4 loss: 0.14962004870176315
Epoch 5 loss: 0.130391463637352
Epoch 6 loss: 0.128837114199996
Epoch 7 loss: 0.14354813192039728
Epoch 8 loss: 0.1360676772892475
Epoch 9 loss: 0.12003622949123383
Epoch 10 loss: 0.12291123159229755
Epoch 11 loss: [illegible]
```



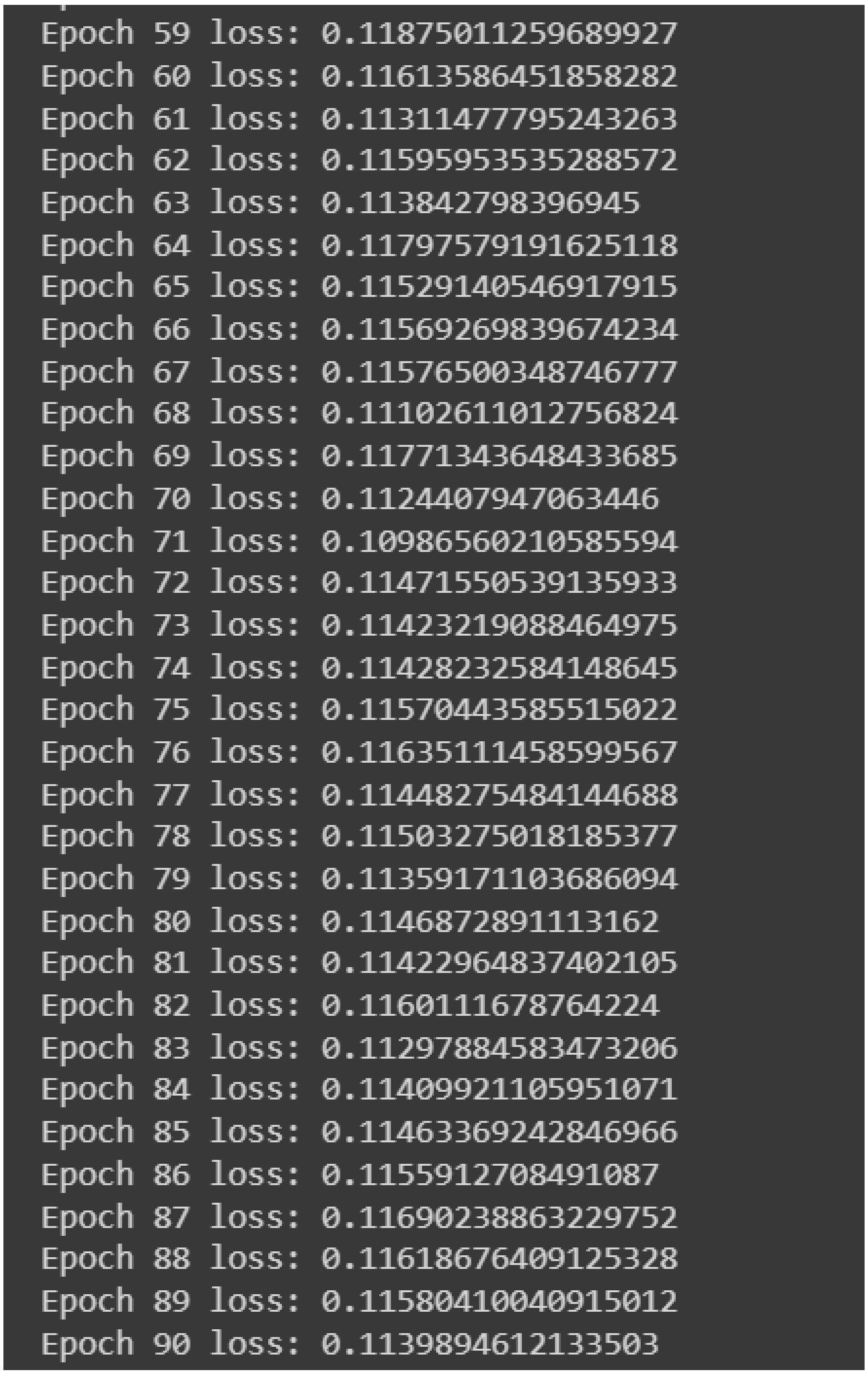
```
Epoch 11 loss: 0.1197291761636734
Epoch 12 loss: 0.1256052851676941
Epoch 13 loss: 0.12614857126027346
Epoch 14 loss: 0.11898323427885771
Epoch 15 loss: 0.12189400754868984
Epoch 16 loss: 0.1229244451969862
Epoch 17 loss: 0.11761567555367947
Epoch 18 loss: 0.12119482085108757
Epoch 19 loss: 0.12213557958602905
Epoch 20 loss: 0.12232185062021017
Epoch 21 loss: 0.12014115694910288
Epoch 22 loss: 0.12230431661009789
Epoch 23 loss: 0.11855146009474993
Epoch 24 loss: 0.11848580371588469
Epoch 25 loss: 0.13054771441966295
Epoch 26 loss: 0.11688564717769623
Epoch 27 loss: 0.12030140683054924
Epoch 28 loss: 0.11841667909175158
Epoch 29 loss: 0.1170450747013092
```



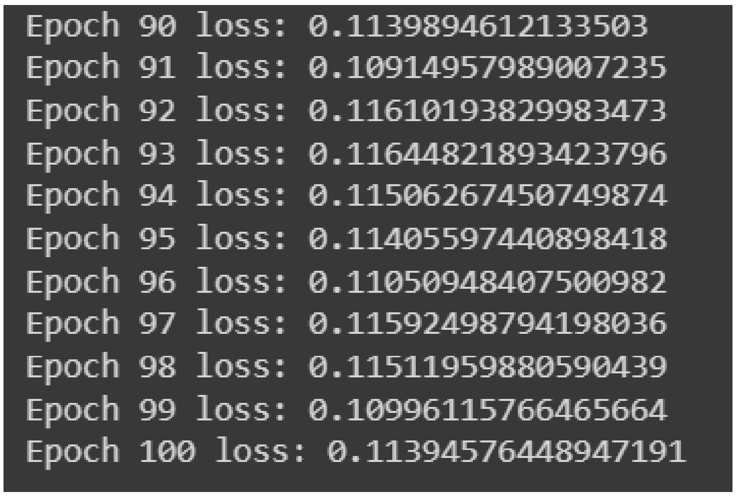
```
 Epoch 30 loss: 0.11297056823968887
 Epoch 31 loss: 0.12114215362817049
 Epoch 32 loss: 0.12216584477573633
 Epoch 33 loss: 0.12051731534302235
 Epoch 34 loss: 0.11818937677890062
 Epoch 35 loss: 0.11907144635915756
 Epoch 36 loss: 0.11672065034508705
 Epoch 37 loss: 0.11759185325354338
 Epoch 38 loss: 0.111037059687078
 Epoch 39 loss: 0.11632787249982357
 Epoch 40 loss: 0.12182456534355879
 Epoch 41 loss: 0.11684448830783367
 Epoch 42 loss: 0.11977221723645926
 Epoch 43 loss: 0.11933332122862339
 Epoch 44 loss: 0.12278787512332201
 Epoch 45 loss: 0.11846382822841406
 Epoch 46 loss: 0.11653306661173701
 Epoch 47 loss: 0.12032783590257168
 Epoch 48 loss: 0.11609573476016521
 Epoch 49 loss: 0.1168378135189414
 Epoch 50 loss: 0.11634654272347689
 Epoch 51 loss: 0.11323279701173306
```



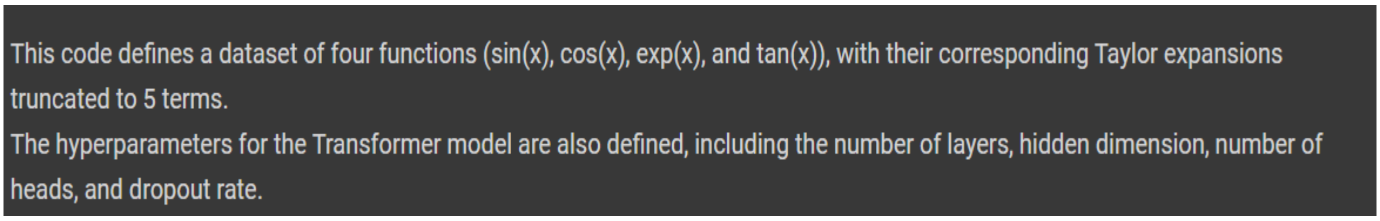
```
Epoch 52 loss: 0.11855622008442879
Epoch 53 loss: 0.11789526604115963
Epoch 54 loss: 0.11410898808389902
Epoch 55 loss: 0.11552436463534832
Epoch 56 loss: 0.11457804497331381
Epoch 57 loss: 0.11612088792026043
Epoch 58 loss: 0.1144281281158328
```



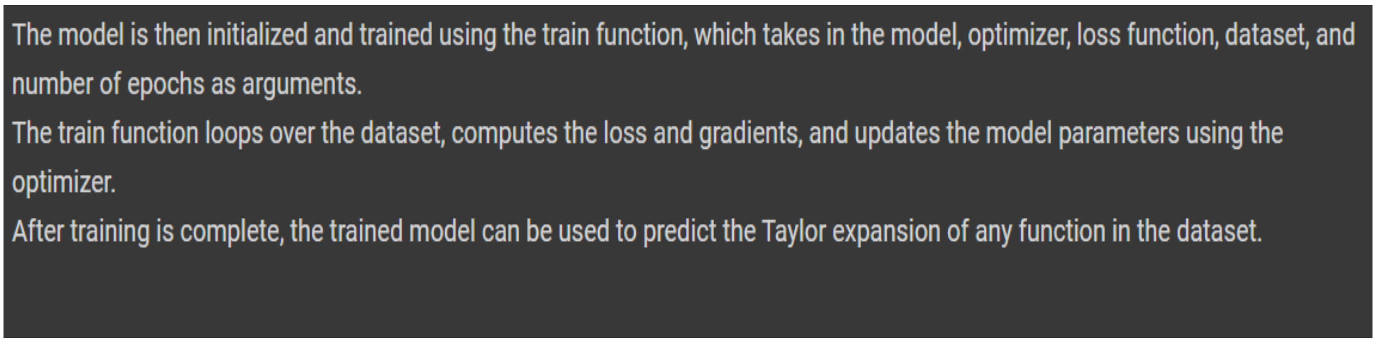
```
Epoch 59 loss: 0.11875011259689927
Epoch 60 loss: 0.11613586451858282
Epoch 61 loss: 0.11311477795243263
Epoch 62 loss: 0.11595953535288572
Epoch 63 loss: 0.113842798396945
Epoch 64 loss: 0.11797579191625118
Epoch 65 loss: 0.11529140546917915
Epoch 66 loss: 0.11569269839674234
Epoch 67 loss: 0.11576500348746777
Epoch 68 loss: 0.11102611012756824
Epoch 69 loss: 0.11771343648433685
Epoch 70 loss: 0.1124407947063446
Epoch 71 loss: 0.10986560210585594
Epoch 72 loss: 0.11471550539135933
Epoch 73 loss: 0.11423219088464975
Epoch 74 loss: 0.11428232584148645
Epoch 75 loss: 0.11570443585515022
Epoch 76 loss: 0.11635111458599567
Epoch 77 loss: 0.11448275484144688
Epoch 78 loss: 0.11503275018185377
Epoch 79 loss: 0.11359171103686094
Epoch 80 loss: 0.1146872891113162
Epoch 81 loss: 0.11422964837402105
Epoch 82 loss: 0.1160111678764224
Epoch 83 loss: 0.11297884583473206
Epoch 84 loss: 0.11409921105951071
Epoch 85 loss: 0.11463369242846966
Epoch 86 loss: 0.1155912708491087
Epoch 87 loss: 0.11690238863229752
Epoch 88 loss: 0.11618676409125328
Epoch 89 loss: 0.11580410040915012
Epoch 90 loss: 0.1139894612133503
```



```
Epoch 90 loss: 0.1139894612133503
Epoch 91 loss: 0.10914957989007235
Epoch 92 loss: 0.11610193829983473
Epoch 93 loss: 0.11644821893423796
Epoch 94 loss: 0.11506267450749874
Epoch 95 loss: 0.11405597440898418
Epoch 96 loss: 0.11050948407500982
Epoch 97 loss: 0.11592498794198036
Epoch 98 loss: 0.11511959880590439
Epoch 99 loss: 0.10996115766465664
Epoch 100 loss: 0.11394576448947191
```



This code defines a dataset of four functions (sin(x), cos(x), exp(x), and tan(x)), with their corresponding Taylor expansions truncated to 5 terms.
The hyperparameters for the Transformer model are also defined, including the number of layers, hidden dimension, number of heads, and dropout rate.



The model is then initialized and trained using the train function, which takes in the model, optimizer, loss function, dataset, and number of epochs as arguments.
The train function loops over the dataset, computes the loss and gradients, and updates the model parameters using the optimizer.
After training is complete, the trained model can be used to predict the Taylor expansion of any function in the dataset.

Colab link :

Task 1:
https://colab.research.google.com/drive/17LRZZyG8vUHCH1PrxwtLNICDPKcHQ13i?usp=sharing

Task 2:
https://colab.research.google.com/drive/1tAvAexircJimMbDkTH0-_QAqCu1n5SVb?usp=sharing

Task 3:
https://colab.research.google.com/drive/1MGZSBqEZ0_DBkM7hkToQnHzfbxaSTYBk?usp=sharing