Certainly! Below is a **minimal self-contained implementation** of the **attention mechanism** in a Transformer, written in pure Python/NumPy, and accompanied by Markdown-formatted explanations.

🧠 Transformer Attention (Self-Attention) in NumPy

📘 Concept

Given a sequence of input vectors $X \in \mathbb{R}^{T \times d}$ where:

- T = sequence length
- d = embedding dimension

The **attention mechanism** computes:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Where:

- $Q = XW^Q$ (queries)
- $K = XW^K$ (keys)
- $V = XW^V$ (values)
- $W^Q, W^K, W^V \in \mathbb{R}^{d \times d_k}$ are learnable weights

👨‍💻 Python Code: Scaled Dot-Product Attention (NumPy)

```python
import numpy as np

def softmax(x, axis=-1):
    x = x - np.max(x, axis=axis, keepdims=True)  # stability
    exp_x = np.exp(x)
    return exp_x / np.sum(exp_x, axis=axis, keepdims=True)

class SelfAttention:
    def __init__(self, embed_dim, head_dim):
        self.embed_dim = embed_dim
        self.head_dim = head_dim

        # Initialize weights: W^Q, W^K, W^V
        self.W_q = np.random.randn(embed_dim, head_dim) * 0.1
        self.W_k = np.random.randn(embed_dim, head_dim) * 0.1
        self.W_v = np.random.randn(embed_dim, head_dim) * 0.1

    def forward(self, X):
        # X: [seq_len, embed_dim]
        Q = X @ self.W_q  # [seq_len, head_dim]
        K = X @ self.W_k
        V = X @ self.W_v

        d_k = self.head_dim
        scores = (Q @ K.T) / np.sqrt(d_k)  # [seq_len, seq_len]
        attn_weights = softmax(scores, axis=-1)  # [seq_len, seq_len]
        output = attn_weights @ V  # [seq_len, head_dim]

        return output, attn_weights
```

📈 Example Usage

```
# Simulate an input sequence of 4 tokens, each with 8-dim embeddings
np.random.seed(42)
X = np.random.randn(4, 8)  # [seq_len=4, embed_dim=8]

attention = SelfAttention(embed_dim=8, head_dim=8)
output, weights = attention.forward(X)

print("Attention Output:\n", output)
print("Attention Weights:\n", weights)
```

📎 Summary (Markdown/LaTeX Recap)

Formula:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

Shape Flow:

- $X \in \mathbb{R}^{T \times d}$
- $Q, K, V \in \mathbb{R}^{T \times d_k}$
- $\text{scores} \in \mathbb{R}^{T \times T}$
- $\text{output} \in \mathbb{R}^{T \times d_k}$

Would you like to extend this to **multi-head attention** or add support for **batch input**?