```
#1. Create Word Embedding
import numpy as np


embeddings = {
    "The": np.array([1.0, 0.0, 0.5]),
    "cat": np.array([0.0, 1.0, 0.3]),
    "sat": np.array([1.0, 1.0, 0.2])
}

# Create the input sequence as a matrix (shape: 3 x 2)
X = np.stack([embeddings["The"], embeddings["cat"], embeddings["sat"]])
```

```
X
```

```
array([[1. , 0. , 0.5],
       [0. , 1. , 0.3],
       [1. , 1. , 0.2]])
```

```
#2. Create Q, K, V Matrices using Learnable Weights
```

```
# Simulated weights (normally learned)
W_Q = np.array([[1.0, 0.0, 0.5], [0.0, 1.0, 0.5],[0.0, 1.0, 0.5]])
W_K = np.array([[1.0, 0.0, 0.5], [0.0, 1.0, 0.5],[0.0, 1.0, 0.5]])
W_V = np.array([[0.5, 1.0, 0.2], [1.0, 0.5, 0.3],[1.0, 0.5, 0.1]])

# Linear projections
Q = X @ W_Q  # (3 x 3)
K = X @ W_K  # (3 x 3)
V = X @ W_V  # (3 x 3)
```

```
Q
```

```
array([[1.  , 0.5 , 0.75],
       [0.  , 1.3 , 0.65],
       [1.  , 1.2 , 1.1 ]])
```

```
K
```

```
array([[1.  , 0.5 , 0.75],
       [0.  , 1.3 , 0.65],
       [1.  , 1.2 , 1.1 ]])
```

```
V
```

```
array([[1.  , 1.25, 0.25],
       [1.3 , 0.65, 0.33],
       [1.7 , 1.6 , 0.52]])
```

```
#3. Compute Dot Products Between Queries and Keys
```

```
scores = Q @ K.T  # (3 x 3)
print("Dot product scores:\n", scores)
```

```
Dot product scores:
 [[1.8125 1.1375 2.425 ]
 [1.1375 2.1125 2.275 ]
 [2.425  2.275  3.65  ]]
```

```
#4. Scale by √d_k
```

```
dk = Q.shape[-1]
scaled_scores = scores / np.sqrt(dk)
```

```
scaled_scores
```

```
array([[1.04644736, 0.65673593, 1.4000744 ],
       [0.65673593, 1.21965244, 1.31347186],
       [1.4000744 , 1.31347186, 2.10732848]])
```

```
#5. Apply Softmax to Get Attention Weights
```

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

```
attention_weights = softmax(scaled_scores)
print("Attention Weights:\n", attention_weights)
```

```
Attention Weights:
 [[0.32242711 0.21836449 0.4592084 ]
 [0.21348029 0.37482567 0.41169404]
 [0.25345618 0.23242983 0.51411399]]
```

```
#6. Multiply by V to Get the Output
```

```
output = attention_weights @ V
print("Final Output:\n", output)
```

```
Final Output:
 [[1.38695523 1.27970424 0.39145543]
 [1.40063353 1.16919751 0.39114344]
 [1.42960874 1.290482   0.40740516]]
```

```
import numpy as np

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

# Step 1: Word embeddings (3 tokens, each with 3-dim embedding)
X = np.array([
    [1.0, 0.0, 1.0],  # "The"
    [0.0, 1.0, 1.0],  # "cat"
    [1.0, 1.0, 0.0]   # "sat"
])  # Shape: (3 tokens x 3 dim)

# Step 2: Use identity matrices for Q, K, V projections (for clarity)
Wq = np.eye(3)  # (3 x 3)
Wk = np.eye(3)
Wv = np.eye(3)

Q = X @ Wq  # (3 x 3)
K = X @ Wk  # (3 x 3)
V = X @ Wv  # (3 x 3)

# Step 3: Compute scaled attention scores
scores = Q @ K.T  # (3 x 3)

# Optional: scale by sqrt(d_k), d_k = 3
dk = Q.shape[-1]
scaled_scores = scores / np.sqrt(dk)

# Step 4: Create causal mask (lower triangular, so we can't see the future)
seq_len = X.shape[0]
mask = np.triu(np.ones((seq_len, seq_len)) * -np.inf, k=1)
```

```
# Apply mask
masked_scores = scaled_scores + mask
```

```
masked_scores
```

```
array([[1.15470054,       -inf,       -inf],
       [0.57735027, 1.15470054,       -inf],
       [0.57735027, 0.57735027, 1.15470054]])
```

```
# Step 5: Softmax over masked scores
attn_weights = softmax(masked_scores)
attn_weights
```

```
array([[1.        , 0.        , 0.        ],
       [0.35954252, 0.64045748, 0.        ],
       [0.26445846, 0.26445846, 0.47108308]])
```

```
# Step 6: Compute attention output
output = attn_weights @ V  # (3 x 3)

# Print everything
print("Q:\n", Q)
print("K:\n", K)
print("V:\n", V)
print("Masked Attention Weights:\n", attn_weights)
print("Masked Self-Attention Output:\n", output)
```

```
Q:
 [[1. 0. 1.]
 [0. 1. 1.]
 [1. 1. 0.]]
K:
 [[1. 0. 1.]
 [0. 1. 1.]
 [1. 1. 0.]]
V:
 [[1. 0. 1.]
 [0. 1. 1.]
 [1. 1. 0.]]
Masked Attention Weights:
 [[1.         0.         0.        ]
 [0.35954252 0.64045748 0.        ]
 [0.26445846 0.26445846 0.47108308]]
Masked Self-Attention Output:
 [[1.         0.         1.        ]
 [0.35954252 0.64045748 1.        ]
 [0.73554154 0.73554154 0.52891692]]
```

Start coding or generate with AI.