Define libraries and sentence

In [11]:

```
import numpy as np
import re
import matplotlib.pyplot as plt
# Define the sentences
sentences = """We are about to study the idea of a computational process.␣
↪Computational processes are abstract beings that inhabit computers.
As they evolve, processes manipulate other abstract things called data.
The evolution of a process is directed by a pattern of rules
called a program. People create programs to direct processes. In effect, we␣
↪conjure the spirits of the computer with our spells."""
```

Data Cleaning

In [12]:

```
# Clean Data
# Remove special characters
sentences = re.sub('[^A-Za-z0-9]+', ' ', sentences)
# Remove 1-letter words
sentences = re.sub(r'\b\w\b', '', sentences).strip()
# Lowercase all characters
sentences = sentences.lower()
```

Data Preparation

In [13]:

```
# Create a list of words and a vocabulary
words = sentences.split()
vocab = set(words)
# Define parameters
vocab_size = len(vocab)
embed_dim = 10
context_size = 2
# Create dictionaries for word indexing
word_to_ix = {word: i for i, word in enumerate(vocab)}
ix_to_word = {i: word for i, word in enumerate(vocab)}
# Prepare data as [(context), target]
data = []
for i in range(context_size, len(words) - context_size):context = [words[i - context_size],
           words[i +context_size]]
target = words[i]
data.append((context, target))
```

generate Training Data

In [14]:

```
# Initialize embeddings
embeddings = np.random.random_sample((vocab_size, embed_dim))
# Linear model
def linear(m, theta):
    w = theta
    return m.dot(w)
# Log softmax and NLL loss
def log_softmax(x):
    e_x = np.exp(x - np.max(x))
    return np.log(e_x / e_x.sum())
def NLLLoss(logs, targets):
    out = logs[range(len(targets)), targets]
    return -out.sum() / len(out)
def log_softmax_crossentropy_with_logits(logits, target):
    out = np.zeros_like(logits)
    out[np.arange(len(logits)), target] = 1
    softmax = np.exp(logits) / np.exp(logits).sum(axis=-1, keepdims=True)
    return (-out + softmax) / logits.shape[0]
# Forward function
def forward(context_idxs, theta):
    m = embeddings[context_idxs].reshape(1, -1)
    n = linear(m, theta)
    o = log_softmax(n)
    return m, n, o
# Backward function
def backward(preds, theta, target_idxs):
    m, n, o = preds
    dlog = log_softmax_crossentropy_with_logits(n, target_idxs)
    dw = m.T.dot(dlog)
    return dw
# Optimize function
def optimize(theta, grad, lr=0.03):
    theta -= grad * lr
    return theta
```

Train Model

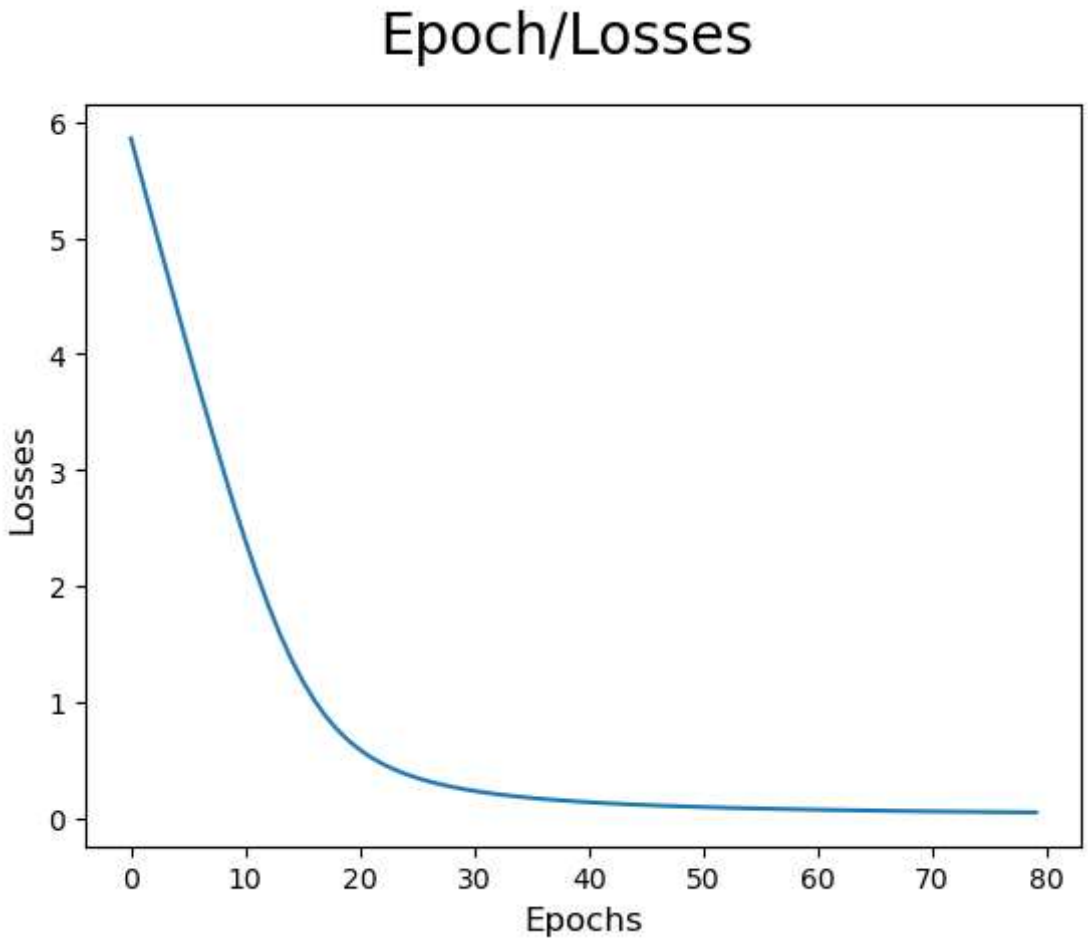
In [15]:

```
# Training the model
theta = np.random.uniform(-1, 1, (2 * context_size * embed_dim, vocab_size))
epoch_losses = {}
for epoch in range(80):
    losses = []
    for context, target in data:
        context_idxs = np.array([word_to_ix[w] for w in context])
        preds = forward(context_idxs, theta)
        target_idxs = np.array([word_to_ix[target]])
        loss = NLLLoss(preds[-1], target_idxs)
        losses.append(loss)
        grad = backward(preds, theta, target_idxs)
        theta = optimize(theta, grad, lr=0.03)
        epoch_losses[epoch] = losses
```

Output

In [16]:

```
# Plot loss/epoch
ix = np.arange(0, 80)
fig = plt.figure()
fig.suptitle('Epoch/Losses', fontsize=20)
plt.plot(ix, [epoch_losses[i][0] for i in ix])
plt.xlabel('Epochs', fontsize=12)
plt.ylabel('Losses', fontsize=12)
plt.show()
# Predict function
def predict(words):
    context_idxs = np.array([word_to_ix[w] for w in words])
    preds = forward(context_idxs, theta)
    word = ix_to_word[np.argmax(preds[-1])]
    return word
```



Example

In [17]:

```
 # Example prediction
print(predict(['we', 'are', 'to', 'study'])) # Example input
# Accuracy function
def accuracy():
    wrong = 0
    for context, target in data:
        if predict(context) != target:
            wrong += 1
        return 1 - (wrong / len(data))
# Calculate accuracy
print(f'Accuracy: {accuracy()}')
```

```
with
Accuracy: 1.0
```

In [ ]: