```
# === Cell 1: Import required libraries ===
import re                                # for regular expressions (used in text cleaning)
import numpy as np                       # for numerical operations and matrix calculations
import string                            # for string operations (optional)
import pandas as pd                      # for handling data (not directly used here)
import matplotlib as mpl                 # for visualizations
import matplotlib.pyplot as plt          # for plotting the word cloud
%matplotlib inline                       # to display plots inline in Jupyter/Colab

from subprocess import check_output      # to execute shell commands (not directly used)
from wordcloud import WordCloud, STOPWORDS # to generate a word cloud visualization
```

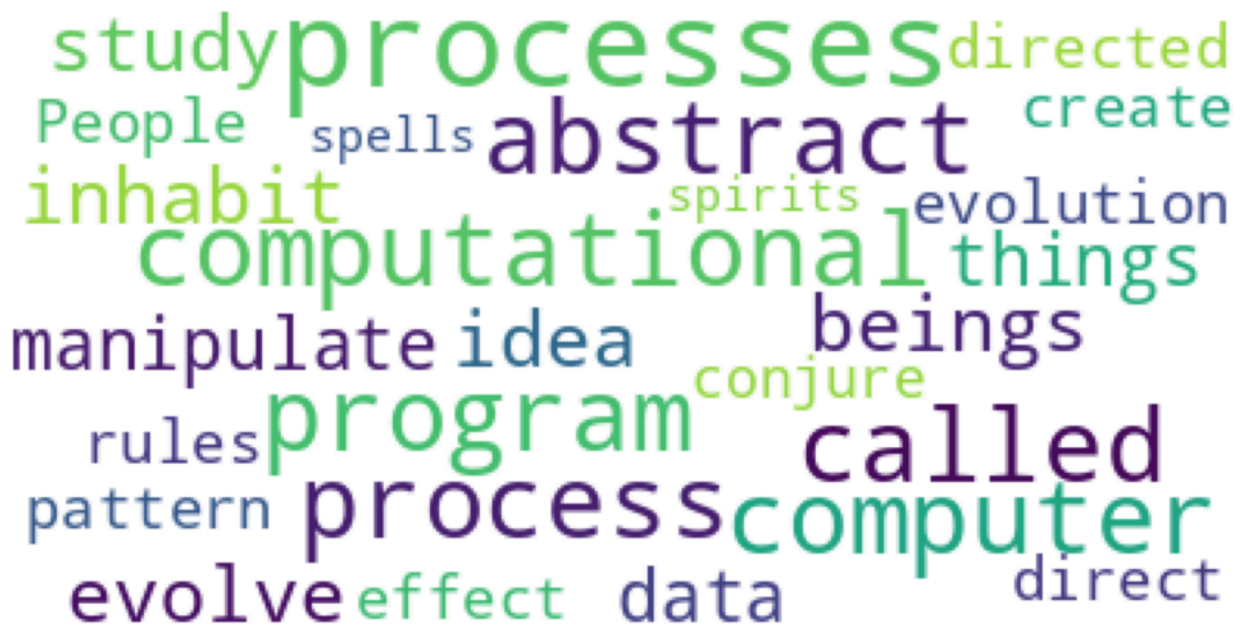
```
UsageError: unrecognized arguments: # to display plots inline in Jupyter/Colab
```

```
# === Cell 2: Initialize text and visualize it ===
stopwords = set(STOPWORDS)               # get default stopwords to ignore common words like 'the', 'is', etc.

# input text (training corpus for CBOW)
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."""

# generate a word cloud from the text
wordcloud = WordCloud(
    background_color='white',            # white background
    stopwords=stopwords,                 # remove common stopwords
    max_words=200,                       # limit number of words
    max_font_size=40,                    # limit font size
    random_state=42                      # random seed for consistent results
).generate(sentences)

# plot the generated word cloud
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(10, 10))  # create a figure
axes.imshow(wordcloud)                   # display the word cloud image
axes.axis('off')                         # turn off axis numbers
fig.tight_layout()                       # adjust layout so image fits neatly
```




```
# === Cell 3: Preprocess text (Data Preparation) ===

# reassign the same text for preprocessing
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."""

# remove special characters using regular expressions
sentences = re.sub('[^A-Za-z0-9]+', ' ', sentences)
```

```
# remove single-letter words (like 'a', 'I') using regex
sentences = re.sub(r'(?:^| )\w(?:$| )', ' ', sentences).strip()

# convert all characters to lowercase for uniformity
sentences = sentences.lower()

# print the cleaned sentence
print(sentences)
```

```
we are about to study the idea of computational process computational processes are abstract beings that inhabit computers a
```

```
# === Cell 4: Tokenize and build vocabulary ===
words = sentences.split()                # split the cleaned text into list of words
vocab = set(words)                       # create a set of unique words (vocabulary)

# print the tokenized words and vocabulary
print(words)
print(vocab)
```

```
['we', 'are', 'about', 'to', 'study', 'the', 'idea', 'of', 'computational', 'process', 'computational', 'processes', 'are',
{'direct', 'we', 'by', 'other', 'create', 'inhabit', 'conjure', 'that', 'with', 'effect', 'evolution', 'rules', 'evolve', 'p
```

```
# === Cell 5: Initialize parameters for CBOW model ===
vocab_size = len(vocab)                  # number of unique words in vocabulary
embed_dim = 10                           # size of embedding vector for each word
context_size = 2                         # number of words to consider on each side of target word

# create word-to-index and index-to-word mappings
word_to_ix = {word: i for i, word in enumerate(vocab)}   # word -> numeric index
ix_to_word = {i: word for i, word in enumerate(vocab)}   # numeric index -> word

# print the dictionaries to check mappings
print(word_to_ix)
print(ix_to_word)
```

```
{'direct': 0, 'we': 1, 'by': 2, 'other': 3, 'create': 4, 'inhabit': 5, 'conjure': 6, 'that': 7, 'with': 8, 'effect': 9, 'evo
{0: 'direct', 1: 'we', 2: 'by', 3: 'other', 4: 'create', 5: 'inhabit', 6: 'conjure', 7: 'that', 8: 'with', 9: 'effect', 10:
```

```
# === Cell 6: Generate training data (context, target pairs) ===
data = []                                # initialize empty list to hold training samples

# loop through each word, skipping first and last two due to context window
for i in range(2, len(words) - 2):
    # context: 2 words before and 2 words after target
    context = [words[i - 2], words[i - 1], words[i + 1], words[i + 2]]
    target = words[i]                    # target word is the middle one
    data.append((context, target))       # append tuple (context, target)

# print first few context-target pairs to verify
print(data[:5])
```

```
[(['we', 'are', 'to', 'study'], 'about'), (['are', 'about', 'study', 'the'], 'to'), (['about', 'to', 'the', 'idea'], 'study'
```

```
# === Cell 7: Initialize embeddings (random start) ===
embeddings = np.random.random_sample((vocab_size, embed_dim))  # random matrix for word embeddings
print(embeddings)                        # print initial embeddings (values between 0 and 1)
```

```
 6.67352298e-01 8.57675316e-01]
 [4.99093669e-01 1.31487265e-01 4.81864060e-01 2.48596269e-02
  7.19785981e-01 7.63927716e-01 1.24407127e-01 5.59845561e-01
  8.09987882e-02 2.53924769e-01]
 [7.08156899e-01 2.43149299e-01 5.67282544e-01 3.90052583e-01
  6.43297297e-01 8.31883642e-01 7.09826387e-01 3.17136409e-01
  9.71533382e-01 8.02672522e-01]
 [6.88032172e-02 5.78914502e-01 3.74562849e-01 2.03550098e-01
  8.08402676e-01 5.52127366e-01 7.92122591e-02 6.55941659e-01
  6.76501390e-01 1.23283644e-01]
 [4.15608497e-01 9.89813201e-01 4.08037597e-01 1.93088910e-01
  2.90975132e-01 2.05053251e-03 8.29147551e-01 1.52374514e-01
  5.03451753e-01 5.85501266e-02]
 [8.29248747e-02 4.30903861e-01 4.18297920e-01 8.07700853e-01
  5.49126536e-02 5.20323953e-01 1.12948268e-01 5.29986036e-01
  4.70463078e-01 5.93562818e-01]
 [6.62541176e-01 3.70753661e-01 8.47686609e-02 1.95098736e-01
  8.38631011e-01 4.63183999e-01 3.49748145e-01 2.74176826e-01
  4.87155384e-01 1.78135772e-01]
 [9.14063380e-01 8.60981296e-01 6.53782396e-01 8.14464618e-01
  7.77740677e-01 8.48961824e-02 9.65719700e-01 2.09681415e-01
  8.47124167e-01 6.04451951e-01]
 [8.90916266e-01 1.84187572e-01 2.44449947e-01 4.49414370e-01
  7.15652157e-01 7.12496846e-01 1.04827004e-01 1.23112775e-02
  1.27488936e-01 2.45193084e-01]
 [9.25459422e-01 7.18184574e-01 3.32716661e-01 3.24501591e-01
  5.21382720e-01 9.49282961e-01 2.28797240e-01 2.08326158e-01
  2.56552114e-03 1.77779056e-01]
 [8.17874999e-01 9.56575891e-01 1.25348389e-01 1.70549654e-01
  5.67145835e-01 4.28498202e-01 5.90067679e-01 9.88894544e-01
  1.63311611e-03 1.30572058e-01]
 [8.01077786e-01 6.64794717e-01 9.31545404e-01 6.28905136e-01
  3.23103093e-02 6.32533795e-01 5.78104252e-01 9.66062914e-01
  1.17671072e-01 3.09049399e-02]
 [6.81018183e-01 8.63907853e-01 4.34492610e-01 5.54540390e-01
  6.12211054e-01 2.85336483e-01 3.32518924e-01 8.54506807e-01
  5.07498840e-01 9.71835518e-01]]
```

```
# === Cell 8: Train CBOW model using NumPy ===
# Initialize weight matrices (these act as trainable parameters)
W1 = np.random.rand(vocab_size, embed_dim)  # input -> hidden weights
W2 = np.random.rand(embed_dim, vocab_size)  # hidden -> output weights

# Set training parameters
learning_rate = 0.01                        # step size for updates
epochs = 1000                               # number of passes over the dataset

# training loop
for epoch in range(epochs):
    loss = 0                                # initialize loss for each epoch
    for context, target in data:            # iterate through each (context, target) pair
        # Convert context and target words to numeric indices
        context_ids = [word_to_ix[w] for w in context]
        target_id = word_to_ix[target]

        # === Forward pass ===
        x = np.mean(W1[context_ids], axis=0)  # average embeddings for context words
        z = np.dot(x, W2)                     # project context vector to output space
        y_pred = np.exp(z) / np.sum(np.exp(z))# softmax: convert to probability distribution

        # === Compute loss ===
        loss -= np.log(y_pred[target_id])     # cross-entropy loss for current prediction

        # === Backpropagation ===
        y_pred[target_id] -= 1                # compute gradient (y_hat - y)
        dW2 = np.outer(x, y_pred)             # gradient for W2
        W2 -= learning_rate * dW2             # update W2 weights

        grad_hidden = np.dot(W2, y_pred)      # gradient for hidden layer (W1 embeddings)
        W1[context_ids] -= learning_rate * grad_hidden / len(context_ids)  # update W1 embeddings

    # print loss every 100 epochs for progress tracking
    if epoch % 100 == 0:
        print(f'Epoch {epoch}, Loss: {loss:.4f}')
```

```
Epoch 0, Loss: 210.2166
Epoch 100, Loss: 168.0740
Epoch 200, Loss: 119.8134
Epoch 300, Loss: 68.3144
Epoch 400, Loss: 34.5065
Epoch 500, Loss: 18.1110
Epoch 600, Loss: 10.7595
Epoch 700, Loss: 7.1643
Epoch 800, Loss: 5.1825
Epoch 900, Loss: 3.9750
```

```
# === Cell 9: Test output - Find similar words ===
print("Similar words to 'process':")       # choose a word to test
similarities = np.dot(W1, W1[word_to_ix['process']])  # compute dot product similarity
sorted_words = np.argsort(-similarities)   # sort indices in descending order of similarity

# print top 5 most similar words
for idx in sorted_words[:5]:
    print(ix_to_word[idx])
```

```
Similar words to 'process':
process
computational
by
computers
they
```

Start coding or generate with AI.