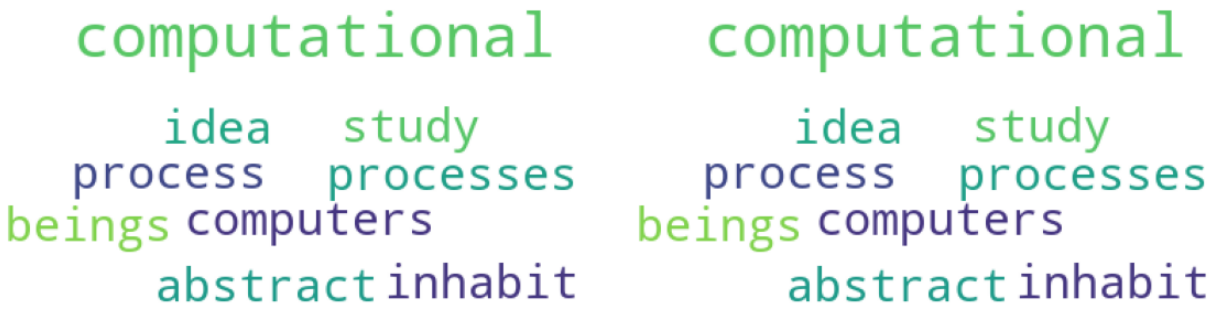
```
import re
import numpy as np
import string
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline
from subprocess import check_output
from wordcloud import WordCloud, STOPWORDS
stopwords = set(STOPWORDS)
data ="""We are about to study the idea of a computational process.
Computational processes are abstract beings that inhabit computers."""
wordcloud = WordCloud(
background_color='white',
stopwords=stopwords,
max_words=200,
max_font_size=40,
random_state=42
).generate(data)
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(24, 24))
axes[0].imshow(wordcloud)
axes[0].axis('off')
axes[1].imshow(wordcloud)
axes[1].axis('off')
axes[2].imshow(wordcloud)
axes[2].axis('off')
fig.tight_layout()
```




```
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
sentences = re.sub('[^A-Za-z0-9]+', ' ', sentences)
# remove 1 letter words
sentences = re.sub(r'(?:^| )\w(?:$| )', ' ', sentences).strip()
```

```
# lower all characters
sentences = sentences.lower()

words = sentences.split()
vocab = set(words)
```

```
vocab_size = len(vocab)
embed_dim = 10
context_size = 2
```

```
word_to_ix = {word: i for i, word in enumerate(vocab)}
ix_to_word = {i: word for i, word in enumerate(vocab)}
```

```
# data - [(context), target]
data = []
for i in range(2, len(words) - 2):
  context = [words[i - 2], words[i - 1], words[i + 1], words[i + 2]]
  target = words[i]
  data.append((context, target))
print(data[:5])
```

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

```
embeddings = np.random.random_sample((vocab_size, embed_dim))
```

```
def linear(m, theta):
  w = theta
  return m.dot(w)
```

```
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]
```

```
def forward(context_idxs, theta):
  m = embeddings[context_idxs].reshape(1, -1)
  n = linear(m, theta)
```

```
  o = log_softmax(n)
  return m, n, o

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
```

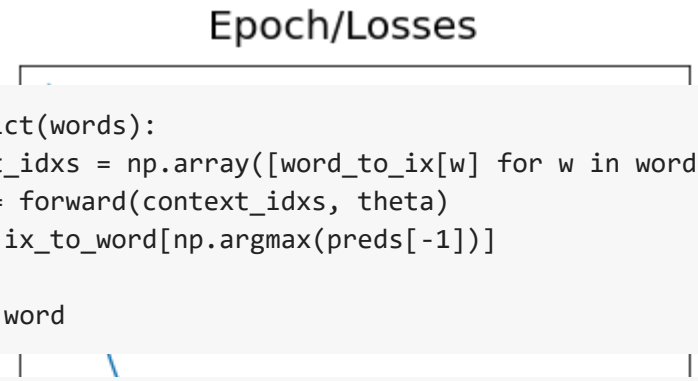
```
def optimize(theta, grad, lr=0.03):
  theta -= grad * lr
  return theta
```

```
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
```

```
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)
```

```
Text(0, 0.5, 'Losses')
```




```
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
```

```
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
```

```
# (['we', 'are', 'to', 'study'], 'about')
predict(['we', 'are', 'to', 'study'])
```

```
'with'
```

```
def accuracy():
 wrong = 0
 for context, target in data:
  if(predict(context) != target):
   wrong += 1
 return (1 - (wrong / len(data)))
```

```
accuracy()
```

```
0.01851851851851849
```

Colab paid products - Cancel contracts here

✓ 0s completed at 11:50 PM