LECTURE 13: CASE STUDY – DATA STRUCTURE SELECTION

Hung-Yu Wei

Case Study: Text Analysis

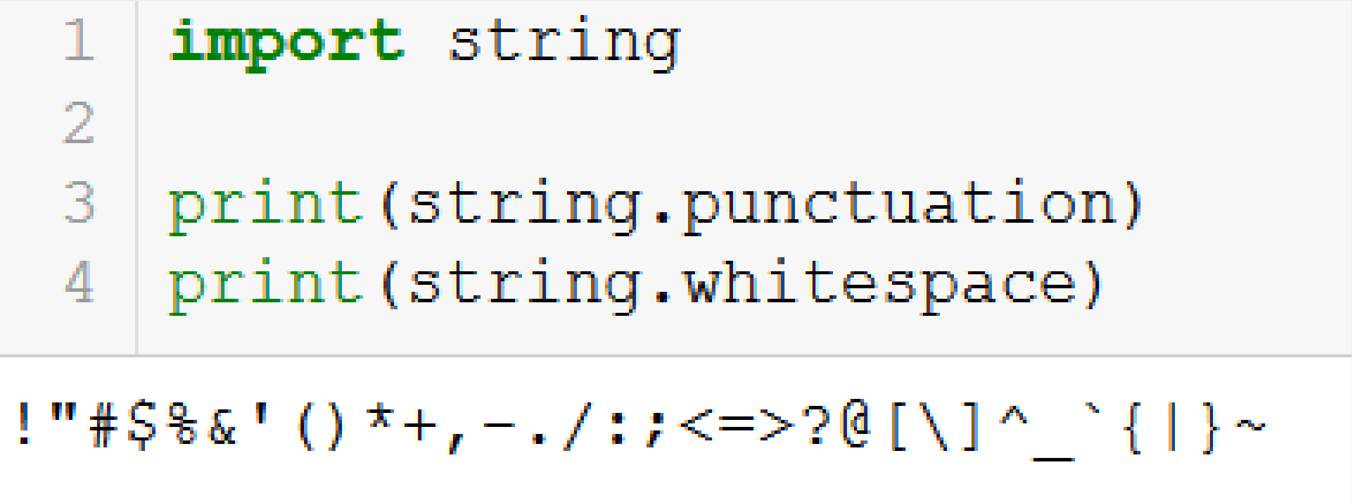
- Read text from a file (e.g. Emma, by Jane Austen)
- Pre-processing
  - *Read line-by-line*
  - *Get rid of punctuation*
    - We are interested in words only
  - *Convert to lower cases*
- Analyze the text
  - *Frequency of words*
    - Histogram
  - *Number of total words*
  - *Words that appear most frequently*
  - *Words that appear in the text but not appearing in another word list*

Extended Case Study: Text Generation

- Generate text randomly
  - *From the previous text analysis*
    - Generate word form the histogram randomly
- More realistic random text generation
  - *Markov analysis*

Syntax: Special strings

- String.punctuation
- String.whitespace

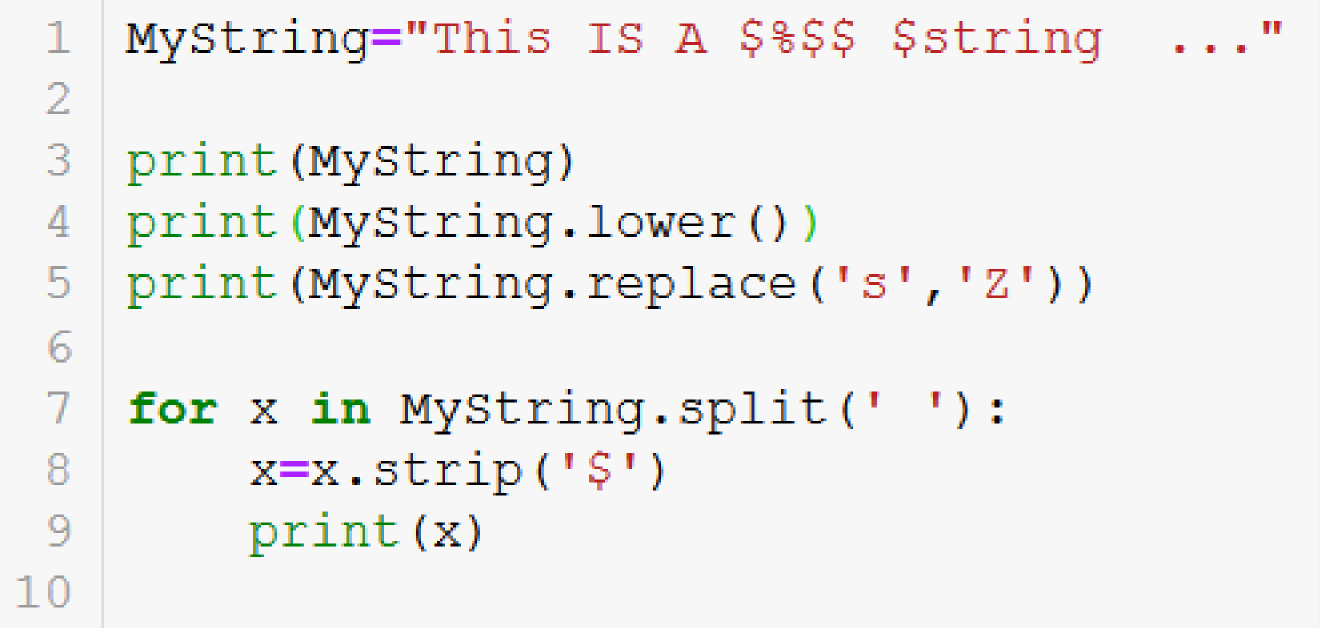


```python
import string

print(string.punctuation)
print(string.whitespace)
```

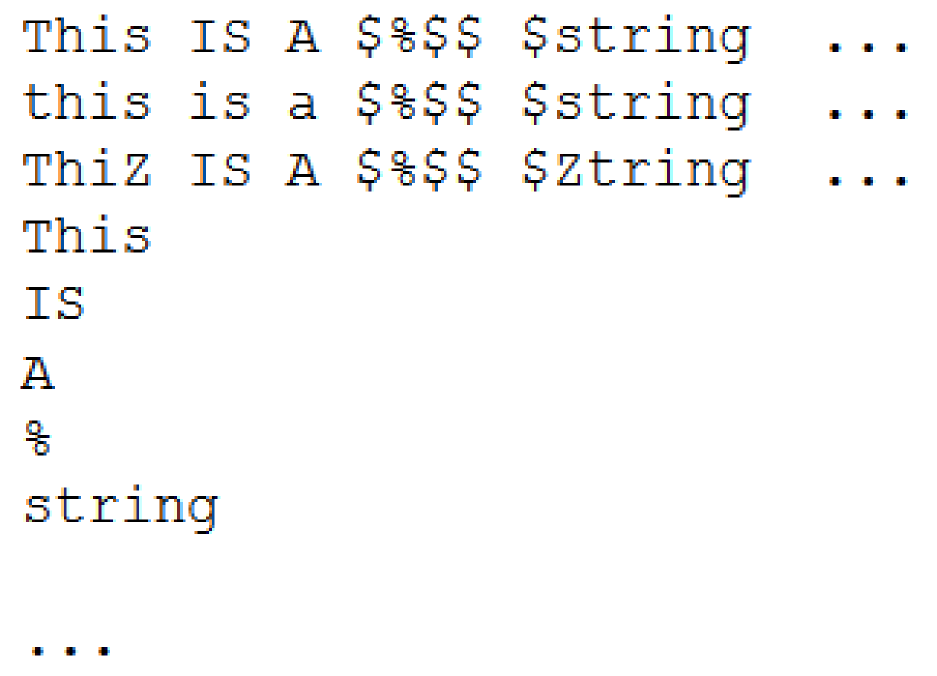
```
!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
```

Syntax: Some string processing

- lower()
- replace()
- split()
- strip()



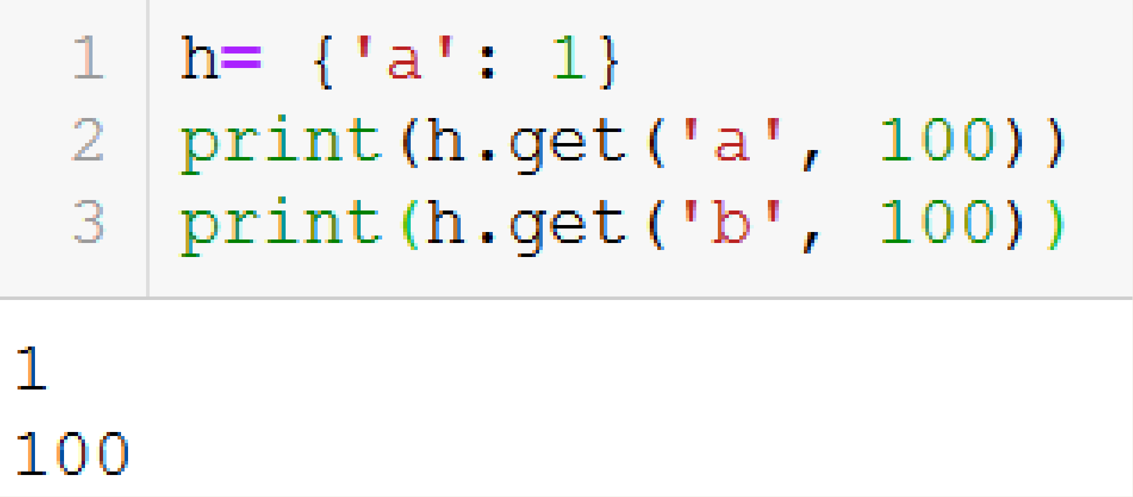
```python
MyString="This IS A $%$$ $string  ..."

print(MyString)
print(MyString.lower())
print(MyString.replace('s','Z'))

for x in MyString.split(' '):
    x=x.strip('$')
    print(x)

```



```
This IS A $%$$ $string  ...
this is a $%$$ $string  ...
ThiZ IS A $%$$ $Ztring  ...
This
IS
A
%
string

...
```

Syntax: Dictionary get()

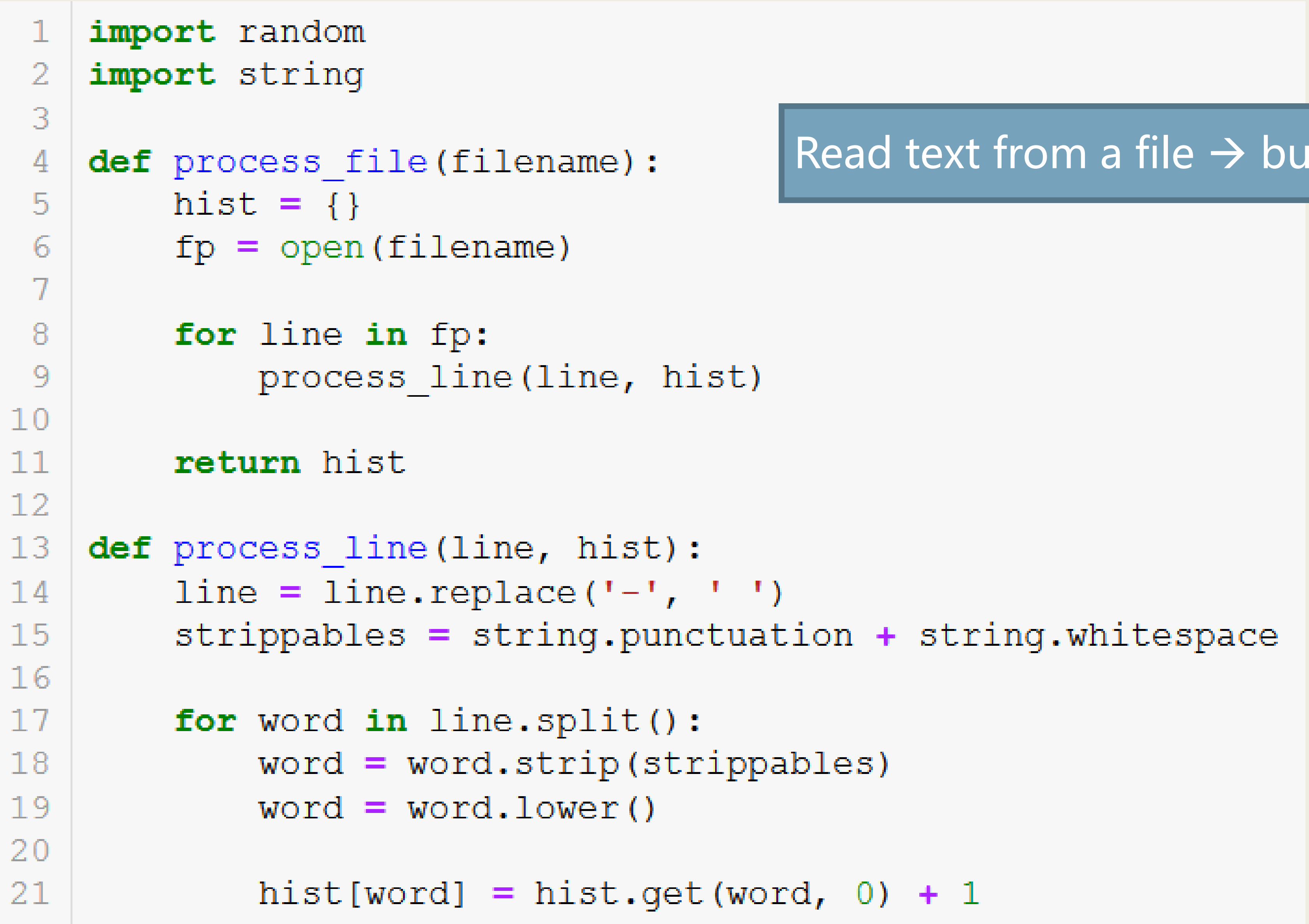
- get that takes a key and a default value.
  - *If the key appears in the dictionary, get returns the corresponding value*
  - *Otherwise it returns the default value.*



```python
h= {'a': 1}
print(h.get('a', 100))
print(h.get('b', 100))
```

```
1
100
```

Read from file and word processing



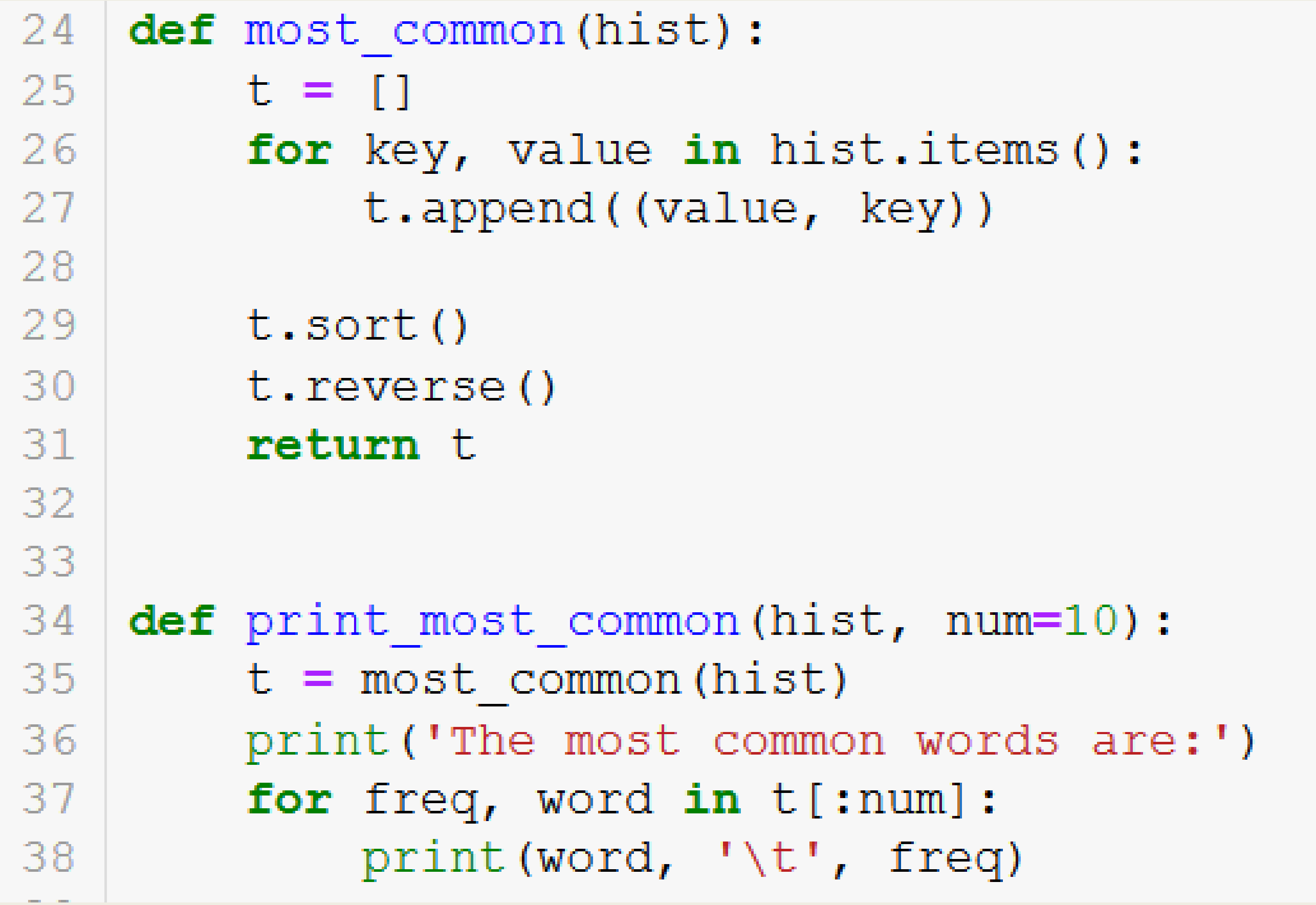
```python
import random
import string

def process_file(filename):
    hist = {}
    fp = open(filename)

    for line in fp:
        process_line(line, hist)

    return hist

def process_line(line, hist):
    line = line.replace('-', ' ')
    strippables = string.punctuation + string.whitespace

    for word in line.split():
        word = word.strip(strippables)
        word = word.lower()

        hist[word] = hist.get(word, 0) + 1
```

Read text from a file → build word histogram

More word processing of a histogram

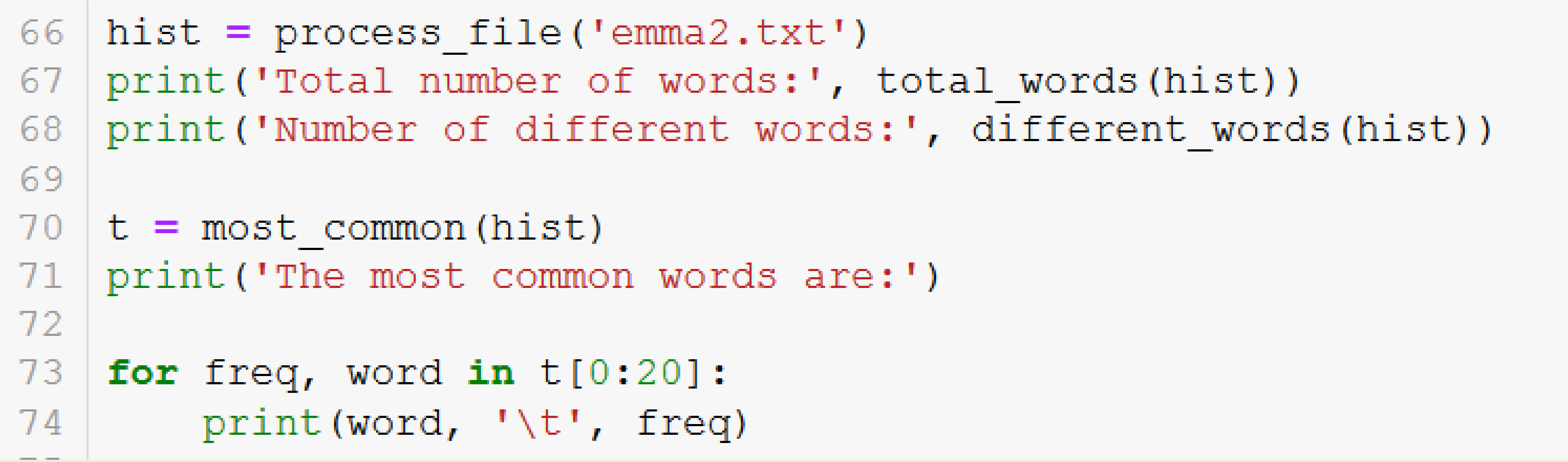
- The total number of words
  - *Add all the frequency values in the histogram*
- The number of different words
  - *The number of items in the dictionary:*
- The most common words
  - *Returns a list of word-frequency tuples*
  - *Sort from high to low*

Most Common word in histogram

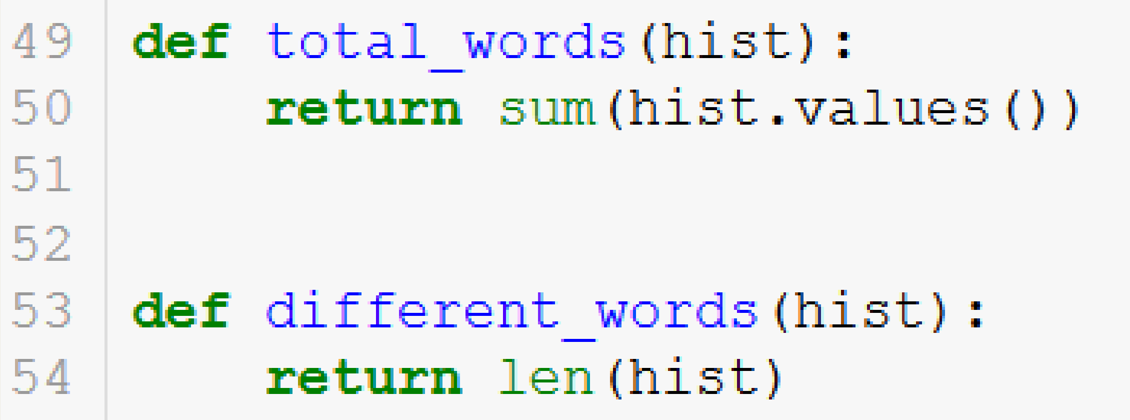


```python
def most_common(hist):
    t = []
    for key, value in hist.items():
        t.append((value, key))

    t.sort()
    t.reverse()
    return t


def print_most_common(hist, num=10):
    t = most_common(hist)
    print('The most common words are:')
    for freq, word in t[:num]:
        print(word, '\t', freq)
```

Total words & different words



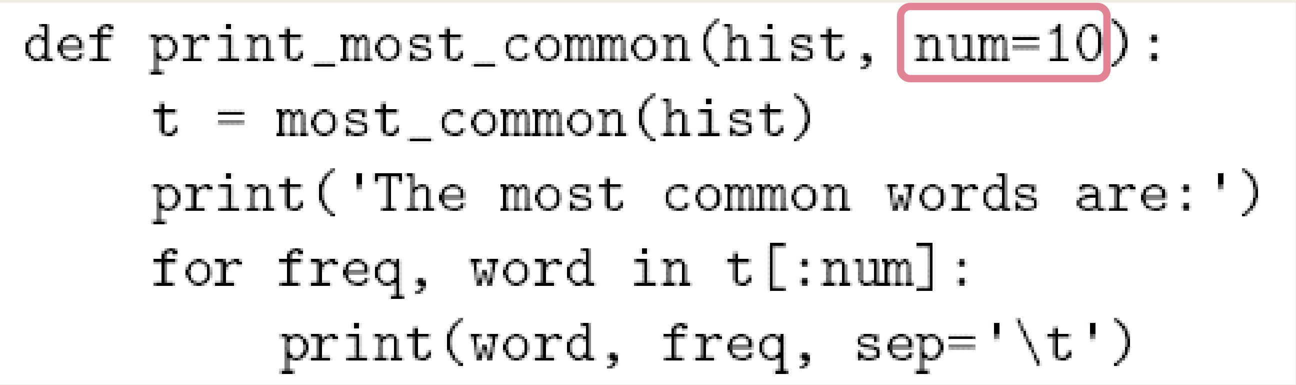
```python
hist = process_file('emma2.txt')
print('Total number of words:', total_words(hist))
print('Number of different words:', different_words(hist))

t = most_common(hist)
print('The most common words are:')

for freq, word in t[0:20]:
    print(word, '\t', freq)
```



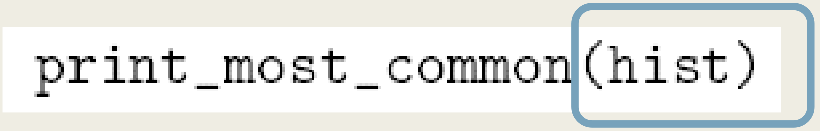
```python
def total_words(hist):
    return sum(hist.values())


def different_words(hist):
    return len(hist)
```

Optional parameters and default value

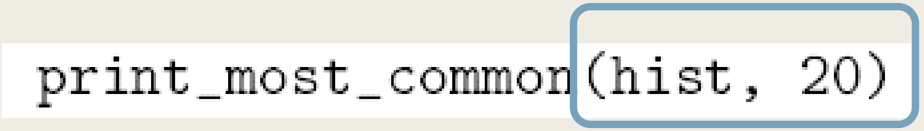
- Functions might
  - *Take optional arguments*
  - *Have default value*
- If there is optional argument, it overrides default value
  - *If a function has both required and optional parameters, all the required parameters have to come first, followed by the optional ones.*



```
def print_most_common(hist, num=10):
    t = most_common(hist)
    print('The most common words are:')
    for freq, word in t[:num]:
        print(word, freq, sep='\t')
```



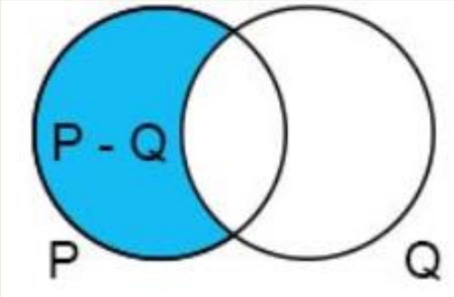
```
print_most_common(hist)
```



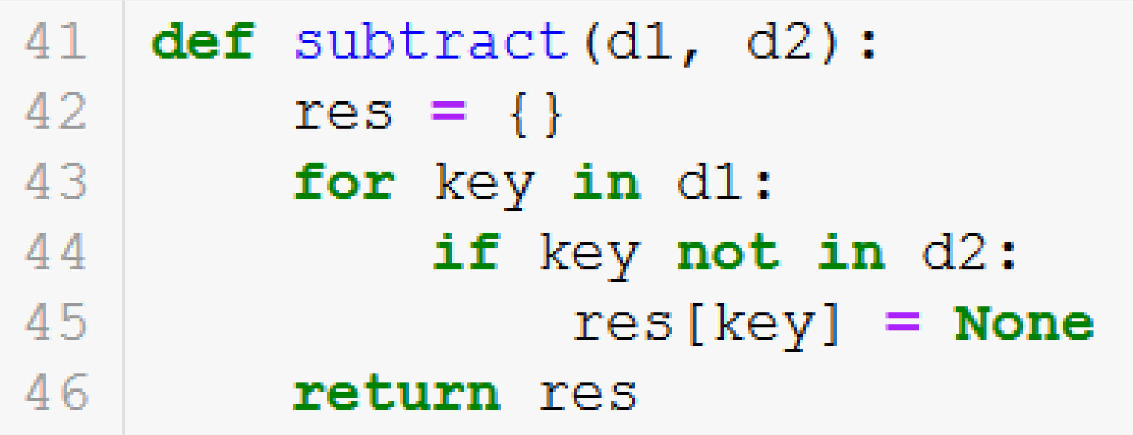
```
print_most_common(hist, 20)
```

Dictionary subtraction

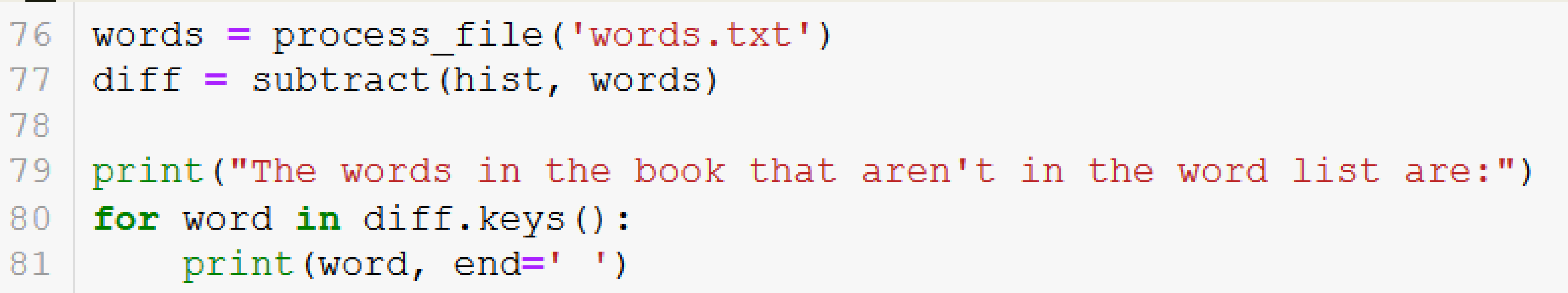
- Finding the words from an article that are not in the existing word list
  - *Set subtraction*
  - *find all the words from one set (the words in the book) that are not in the other (the words in the list)*



Subtract: text not in the word list



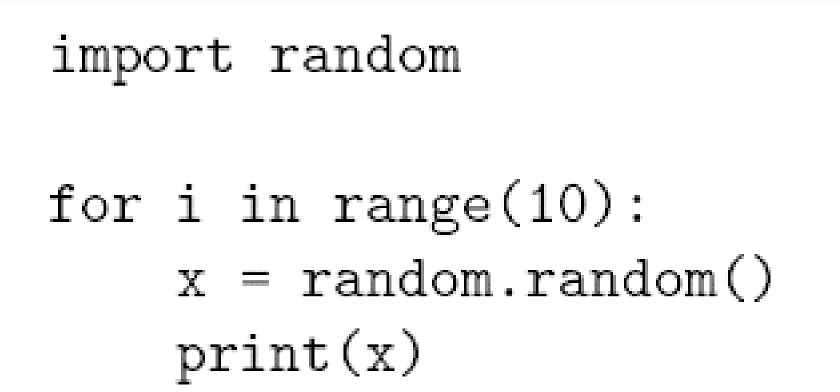
```python
def subtract(d1, d2):
    res = {}
    for key in d1:
        if key not in d2:
            res[key] = None
    return res
```



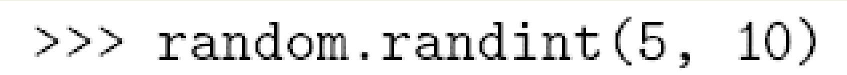
```python
words = process_file('words.txt')
diff = subtract(hist, words)

print("The words in the book that aren't in the word list are:")
for word in diff.keys():
    print(word, end=' ')
```

Syntax: Random Number Generation

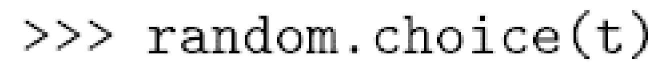
- Created with pseudo-random sequence
- Random module
  - *import random*
- Random functions
  - *random()*
    - Real number between 0.0~1.0
  - *randint(low_num,high_num)*
    - Integer between low_num ~ high_num
  - *choice()*
    - Randomly choose from a sequence



```
import random

for i in range(10):
    x = random.random()
    print(x)
```



```
>>> random.randint(5, 10)
```

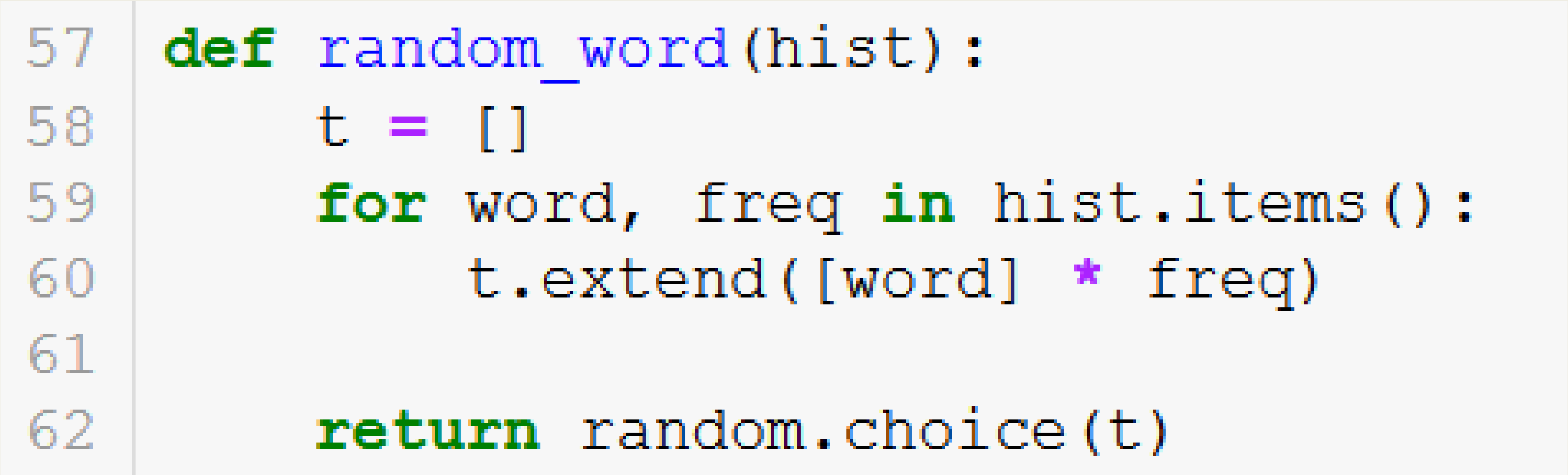


```
>>> random.choice(t)
```

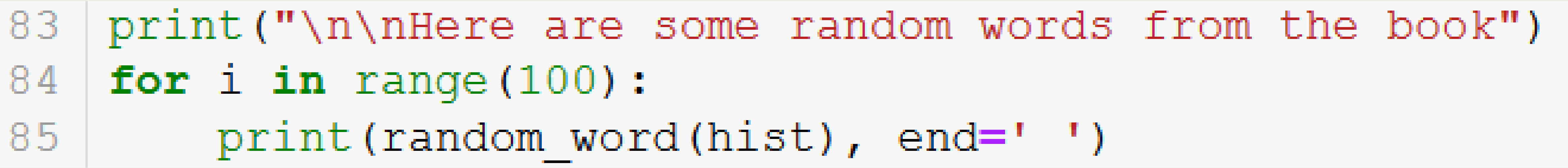
Random words

- Random word generator based on histogram
  - *Basic*
    - Create long word list (a word repeats multiple times)
  - *Slightly improved*
    - Only create the list once
  - *Lower the complexity (Exercise)*
    - 1. Use keys to get a list of the words in the book.
    - 2. Build a list that contains the cumulative sum of the word frequencies . The last item in this list is the total number of words in the book, n.
    - 3. Choose a random number from 1 to n. Use a bisection search (binary search) to find the index where the random number would be inserted in the cumulative sum.
    - 4. Use the index to find the corresponding word in the word list.

Random Words



```python
def random_word(hist):
    t = []
    for word, freq in hist.items():
        t.extend([word] * freq)

    return random.choice(t)
```



```python
print("\n\nHere are some random words from the book")
for i in range(100):
    print(random_word(hist), end=' ')
```

Markov Analysis

- Conditional probability
  - *The probability of the words that might come next.*
- Creating more meaningful text
  - *Dictionary that maps from prefixes to possible suffixes*

Data structure selection

- Select a “suitable” data structure for your implementation
  - *Easy to implement*
  - *Low computational complexity*
  - *Low memory usage*

- Benchmarking
  - *Profile module*
    - https://docs.python.org/3/library/profile.html#introduction-to-the-profilers

Reading

- Chapter 13 in textbook “Think Python”