University of Michigan - Python for Informatics

Notes written by: Abdulrahman AlQallaf

Last modification date: 27-Jun-2016

Remark: These notes are primarily written after completing the following courses offered by the University of Michigan on Coursera:

- Programming for Everybody (Getting Started with Python)
- Python Data Structures
- Using Python to Access Web Data
- Using Databases with Python

I have incorporated information from other sources in some sections to help me better understand the material. These notes are based on my understanding and they are intended to be used as a summary for me to go back to from time to time. Anyone reading this should not rely on my summary and should always revert back to the original sources.

Chapter 1: Why should you learn to write programs?

- Python is a widely used high-level, general-purpose, interpreted, dynamic programming language.
- Python is case sensitive
- To exit the Python interactive mode, type quit()
- Python scripts end in ".py" suffix
- Types of errors
  - Syntax errors
  - Logic errors (e.g., order of operation)
  - Semantic errors
- Python reserved words
  - and
  - del
  - from
  - not
  - while
  - as
  - elif
  - global
  - or
  - with
  - assert
  - else
  - if
  - pass
  - yield
  - break
  - except

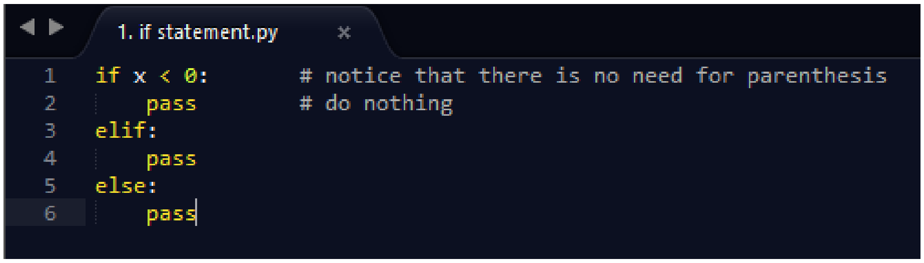
- import
- print
- class
- exec
- in
- raise
- continue
- finally
- is
- return
- def
- for
- lambda
- try

Chapter 2: Variables, expressions, and statements

- Rules for naming variables
  - Alphanumeric
  - Cannot start with a number
  - Can have underscore
  - @ is an illegal character
- To know the type of a variable
  - type(myVariableName)
- An assignment statement creates a new variable and gives it a value
- To display the value of a variable
  - Print myVariableName
- Python (2.x) uses floor division when applied to integers
- Order of operations → PEMDAS
  - Parenthesis
  - Exponentiation
  - Multiplication and division
  - Addition and subtraction
  - Operators with the same precedence are evaluated from left to right
- Modulus operator → %
  - To get the right-most digit, use x % 10
- The + operator concatenates strings
- To get user input, use the following
  - Raw_input('text you want to diplay at prompt')
- There is no difference between ' and "
- To convert the type of a variable
  - To integer → int(myVariableName)
  - To float → float(x)
  - To string → str()
- Comments
  - Single line → # my comment
  - Multi-line → ''' my comment '''

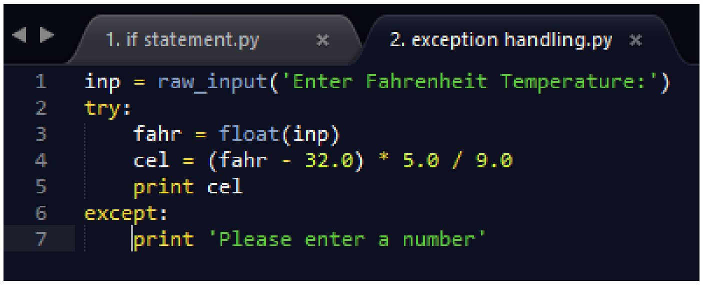
Chapter 3: Conditional execution

- Comparison operators
  - x == y
  - x != y
  - x <= y
  - x >= y
  - x < y
  - x > y
  - x is y   # x is referring to the same variable (i.e., same memory location) as y
  - x is not y
- Logical operators
  - and
  - or
  - not
  - ➔ any nonzero number is interpreted as true
- If statement



```
if x < 0:        # notice that there is no need for parenthesis
    pass         # do nothing
elif:
    pass
else:
    pass
```

- Handling exceptions

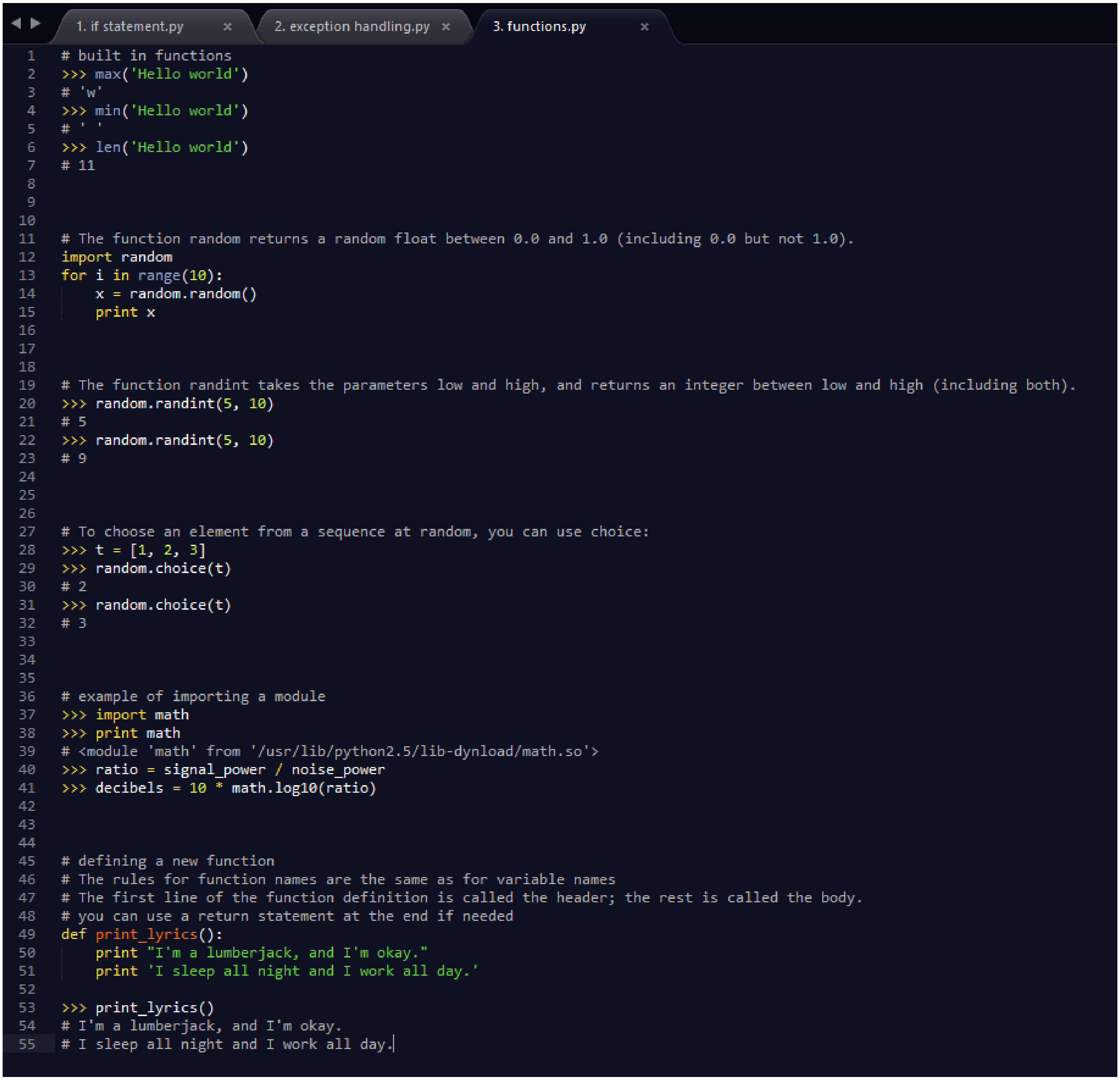


```
inp = raw_input('Enter Fahrenheit Temperature:')
try:
    fahr = float(inp)
    cel = (fahr - 32.0) * 5.0 / 9.0
    print cel
except:
    print 'Please enter a number'
```

- For logical expressions, Python does short-circuit evaluation (i.e., it evaluates a logical expression from left to right and stops the evaluation process when it figured out the final answer)
- compound statement: A statement that consists of a header and a body. The header ends with a colon “:”. The body is indented relative to the header.

Chapter 4: Functions

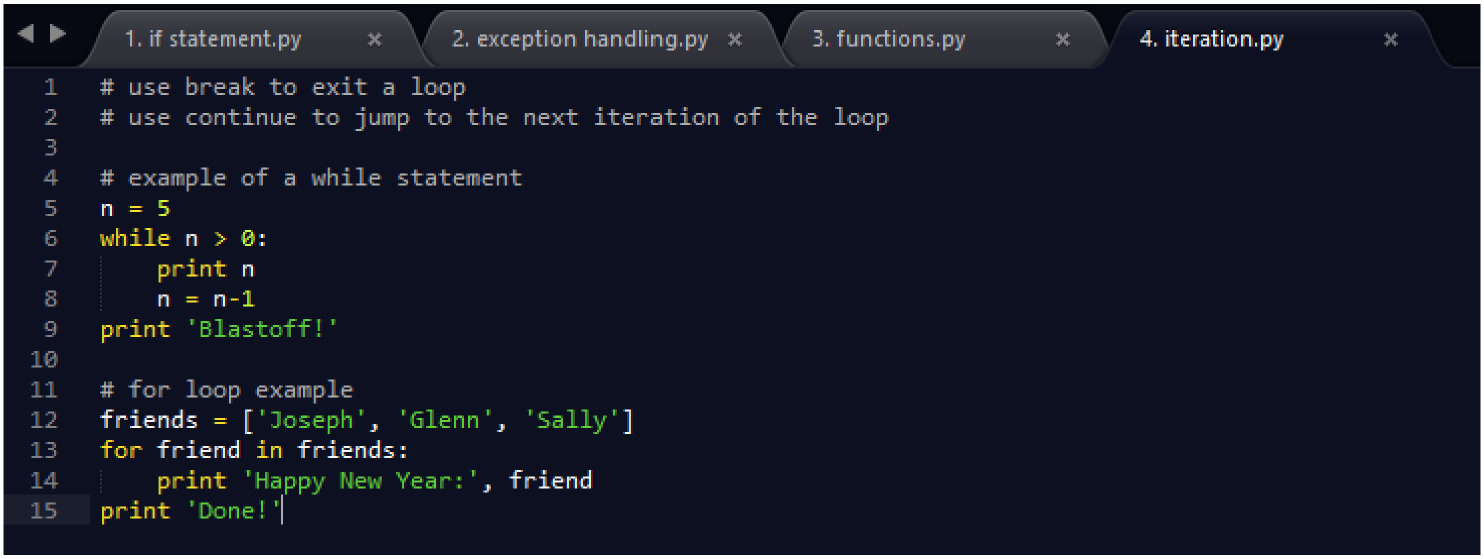
- In Python, Object references are passed by value (basically passing a copy of the address to the object)



```
# built in functions
>>> max('Hello world')
# 'w'
>>> min('Hello world')
# ' '
>>> len('Hello world')
# 11



# The function random returns a random float between 0.0 and 1.0 (including 0.0 but not 1.0).
import random
for i in range(10):
    x = random.random()
    print x



# The function randint takes the parameters low and high, and returns an integer between low and high (including both).
>>> random.randint(5, 10)
# 5
>>> random.randint(5, 10)
# 9



# To choose an element from a sequence at random, you can use choice:
>>> t = [1, 2, 3]
>>> random.choice(t)
# 2
>>> random.choice(t)
# 3



# example of importing a module
>>> import math
>>> print math
# <module 'math' from '/usr/lib/python2.5/lib-dynload/math.so'>
>>> ratio = signal_power / noise_power
>>> decibels = 10 * math.log10(ratio)



# defining a new function
# The rules for function names are the same as for variable names
# The first line of the function definition is called the header; the rest is called the body.
# you can use a return statement at the end if needed
def print_lyrics():
    print "I'm a lumberjack, and I'm okay."
    print 'I sleep all night and I work all day.'

>>> print_lyrics()
# I'm a lumberjack, and I'm okay.
# I sleep all night and I work all day.
```

Chapter 5: Iteration

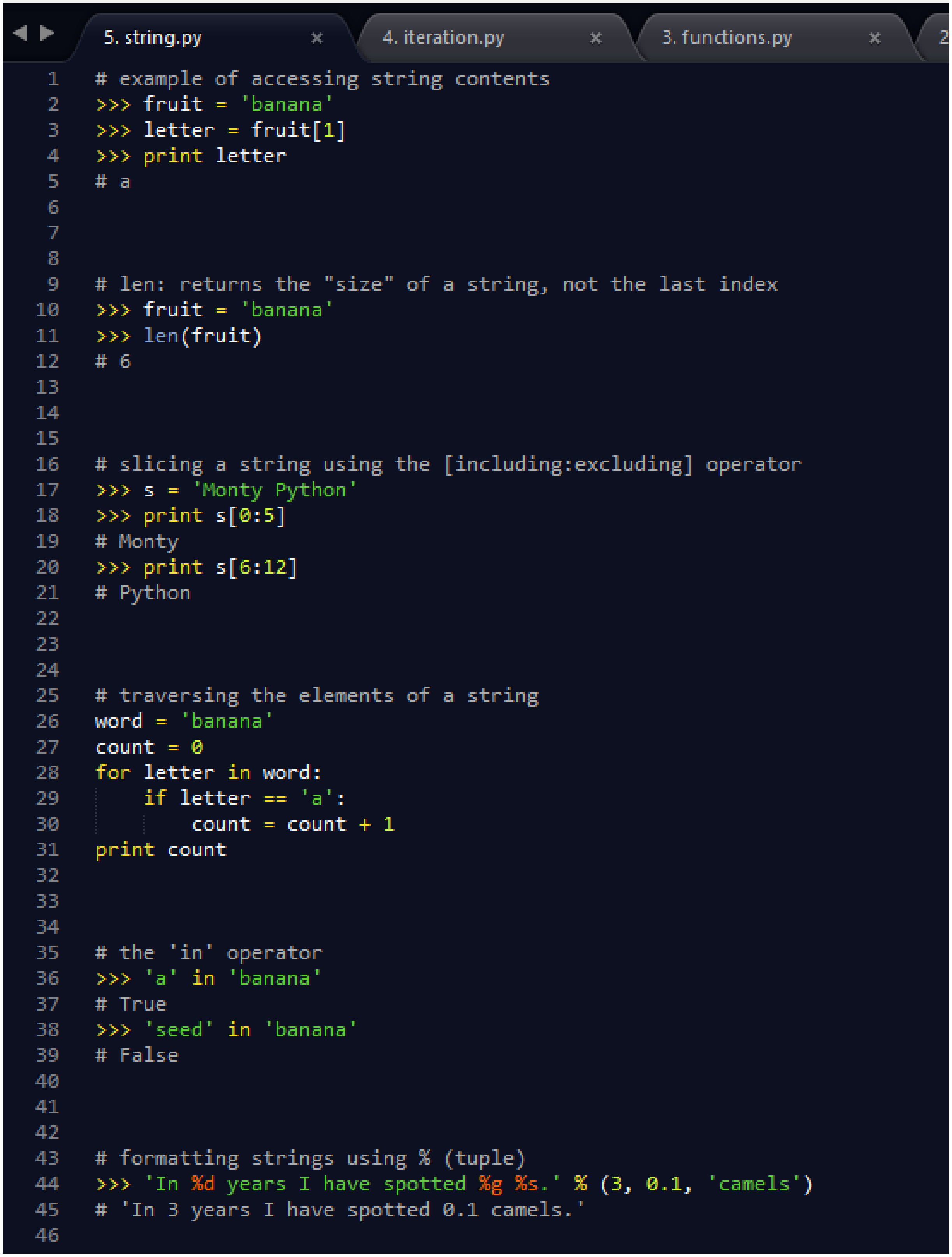
- Instead of NULL, Python uses None



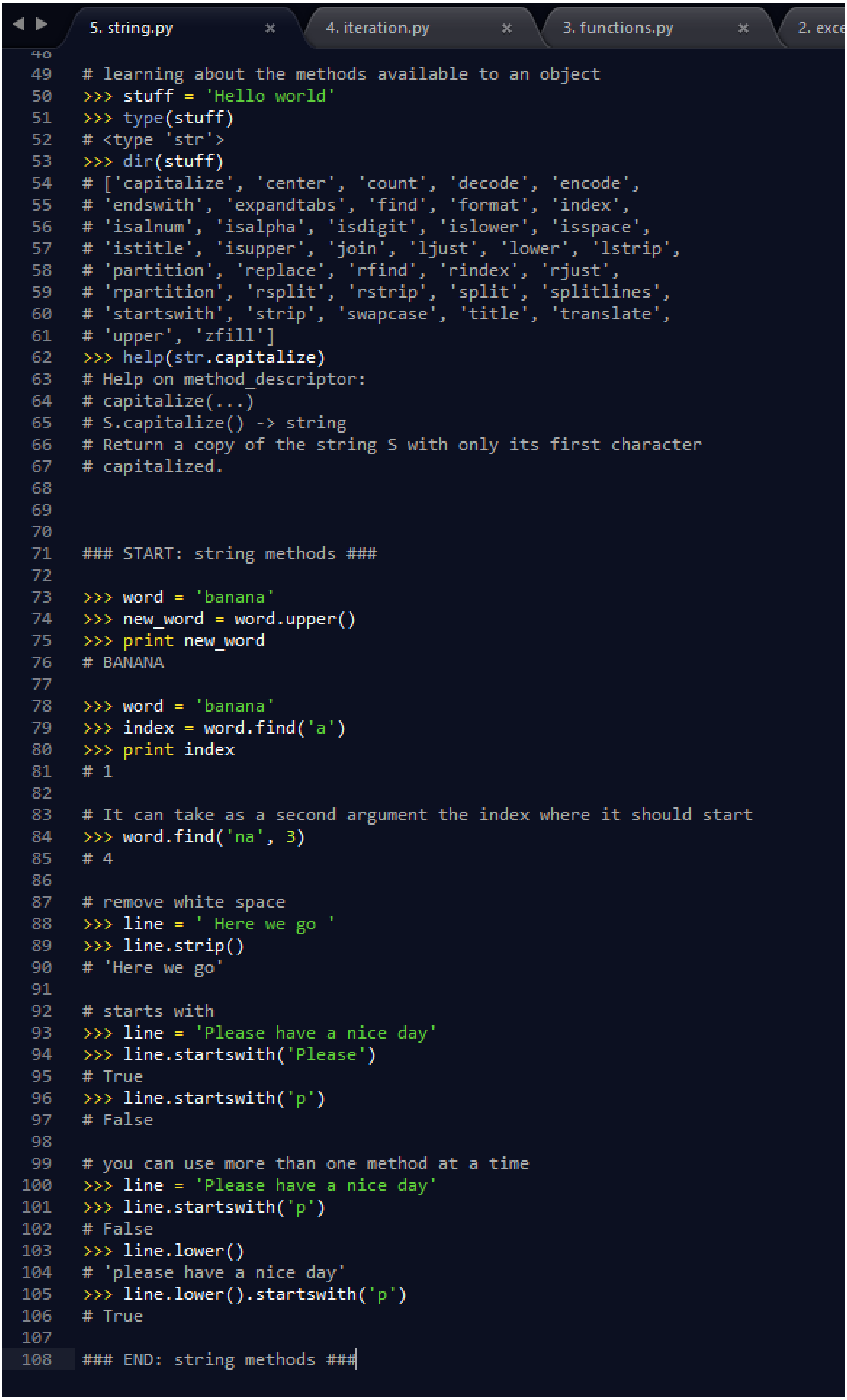
```python
# use break to exit a loop
# use continue to jump to the next iteration of the loop

# example of a while statement
n = 5
while n > 0:
    print n
    n = n-1
print 'Blastoff!'

# for loop example
friends = ['Joseph', 'Glenn', 'Sally']
for friend in friends:
    print 'Happy New Year:', friend
print 'Done!'
```

Chapter 6: Strings

- There are two types of objects in Python
    - Mutable objects
    - Immutable objects → strings
- You can compare strings using the comparison operators



```
# example of accessing string contents
>>> fruit = 'banana'
>>> letter = fruit[1]
>>> print letter
# a



# len: returns the "size" of a string, not the last index
>>> fruit = 'banana'
>>> len(fruit)
# 6



# slicing a string using the [including:excluding] operator
>>> s = 'Monty Python'
>>> print s[0:5]
# Monty
>>> print s[6:12]
# Python



# traversing the elements of a string
word = 'banana'
count = 0
for letter in word:
    if letter == 'a':
        count = count + 1
print count



# the 'in' operator
>>> 'a' in 'banana'
# True
>>> 'seed' in 'banana'
# False



# formatting strings using % (tuple)
>>> 'In %d years I have spotted %g %s.' % (3, 0.1, 'camels')
# 'In 3 years I have spotted 0.1 camels.'

```



```
# learning about the methods available to an object
>>> stuff = 'Hello world'
>>> type(stuff)
# <type 'str'>
>>> dir(stuff)
# ['capitalize', 'center', 'count', 'decode', 'encode',
# 'endswith', 'expandtabs', 'find', 'format', 'index',
# 'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace',
# 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip',
# 'partition', 'replace', 'rfind', 'rindex', 'rjust',
# 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines',
# 'startswith', 'strip', 'swapcase', 'title', 'translate',
# 'upper', 'zfill']
>>> help(str.capitalize)
# Help on method_descriptor:
# capitalize(...)
# S.capitalize() -> string
# Return a copy of the string S with only its first character
# capitalized.



### START: string methods ###

>>> word = 'banana'
>>> new_word = word.upper()
>>> print new_word
# BANANA

>>> word = 'banana'
>>> index = word.find('a')
>>> print index
# 1

# It can take as a second argument the index where it should start
>>> word.find('na', 3)
# 4

# remove white space
>>> line = ' Here we go '
>>> line.strip()
# 'Here we go'

# starts with
>>> line = 'Please have a nice day'
>>> line.startswith('Please')
# True
>>> line.startswith('p')
# False

# you can use more than one method at a time
>>> line = 'Please have a nice day'
>>> line.startswith('p')
# False
>>> line.lower()
# 'please have a nice day'
>>> line.lower().startswith('p')
# True

### END: string methods ###
```

Chapter 7: Files

The *open* Function (refer to http://www.tutorialspoint.com/python/python_files_io.htm for the full article)

Before you can read or write a file, you have to open it using Python's built-in *open()* function. This function creates a **file** object, which would be utilized to call other support methods associated with it.

Syntax

```
file object = open(file_name [, access_mode][, buffering])
```

Here are parameter details:

- **file_name:** The file_name argument is a string value that contains the name of the file that you want to access.
- **access_mode:** The access_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc. A complete list of possible values is given below in the table. This is optional parameter and the default file access mode is read (r).
- **buffering:** If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default(default behavior).

Here is a list of the different modes of opening a file:

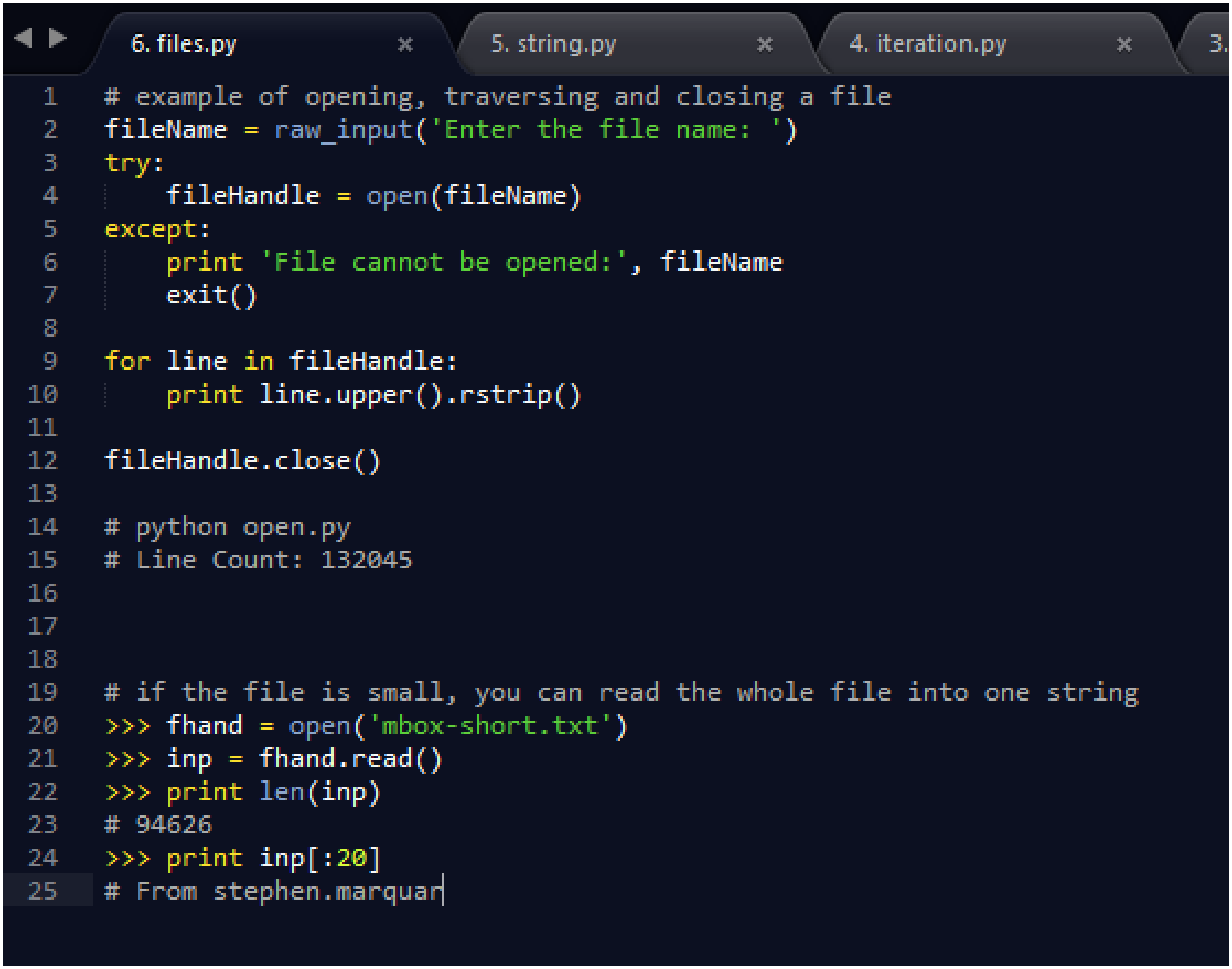
Modes	Description
r	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
rb	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
r+	Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
rb+	Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
wb	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
wb+	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
a	Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
ab	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
a+	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
ab+	Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

The *file* Object Attributes

Once a file is opened and you have one *file* object, you can get various information related to that file.
Here is a list of all attributes related to file object:

Attribute	Description
file.closed	Returns true if file is closed, false otherwise.
file.mode	Returns access mode with which file was opened.
file.name	Returns name of the file.
file.softspace	Returns false if space explicitly required with print, true otherwise.

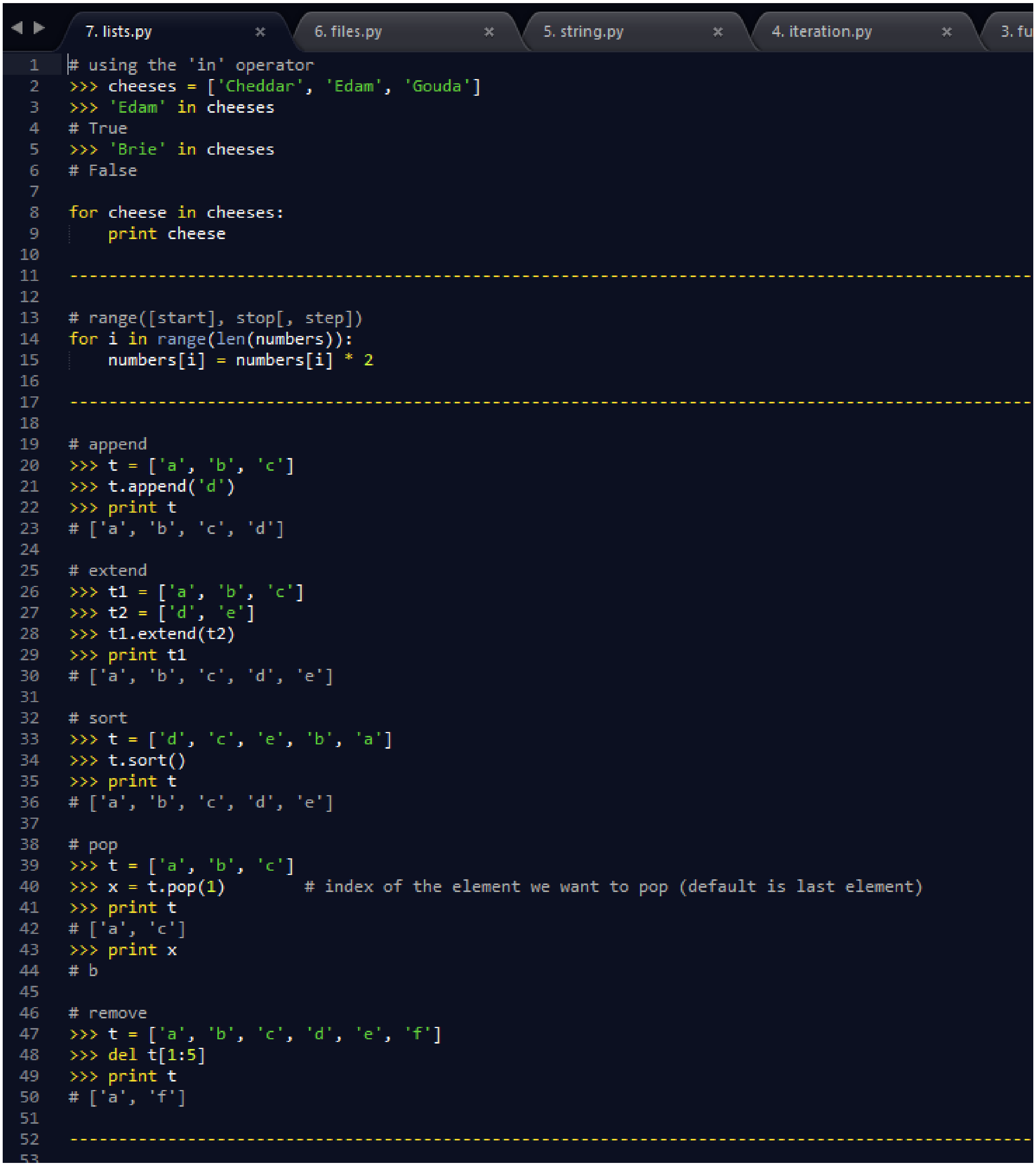
- If the file already exists, opening it in write mode clears out the old data and starts fresh, so be careful! If the file doesn’t exist, a new one is created.



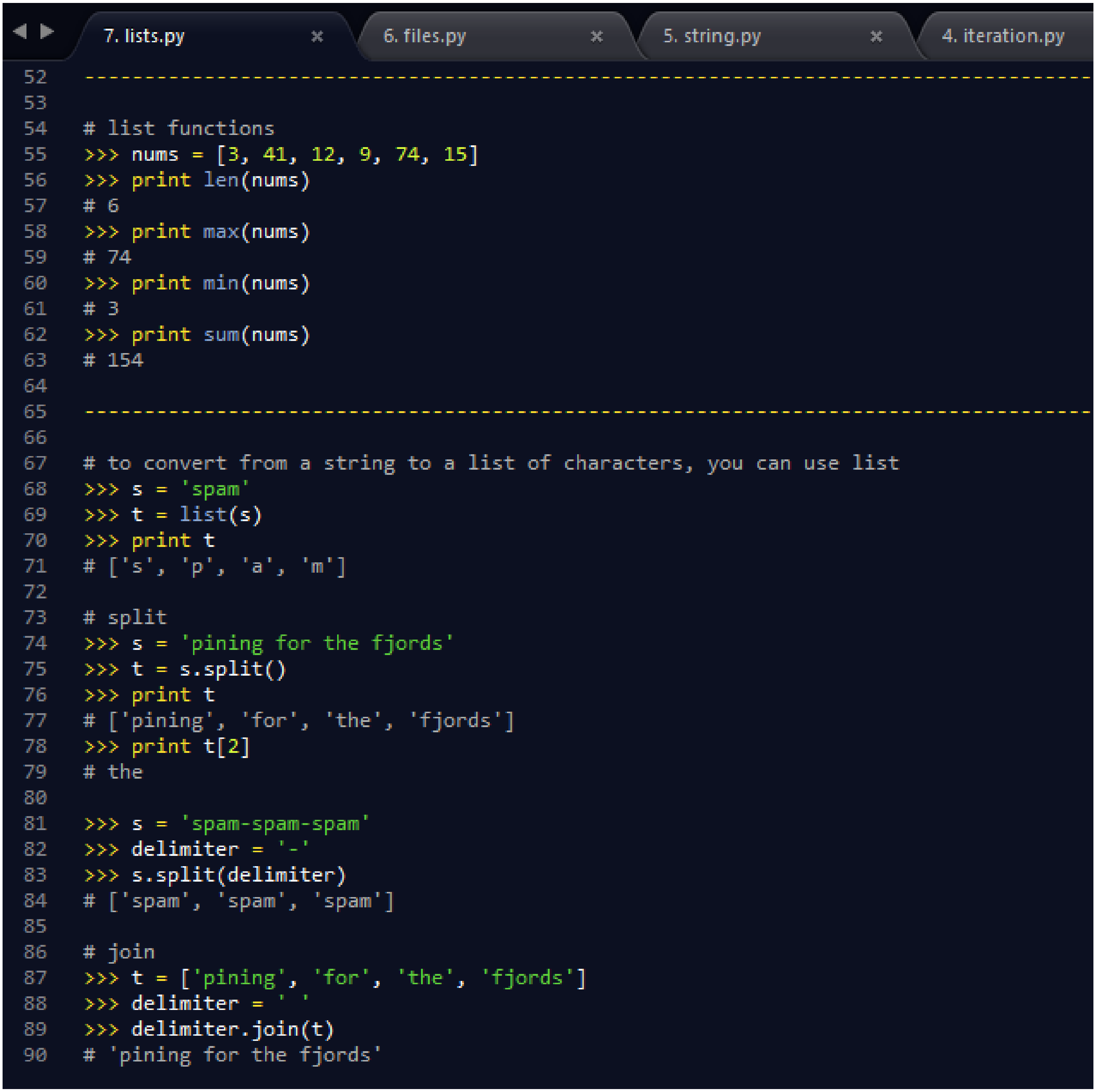
```
# example of opening, traversing and closing a file
fileName = raw_input('Enter the file name: ')
try:
    fileHandle = open(fileName)
except:
    print 'File cannot be opened:', fileName
    exit()

for line in fileHandle:
    print line.upper().rstrip()

fileHandle.close()

# python open.py
# Line Count: 132045



# if the file is small, you can read the whole file into one string
>>> fhand = open('mbox-short.txt')
>>> inp = fhand.read()
>>> print len(inp)
# 94626
>>> print inp[:20]
# From stephen.marquar
```

Chapter 8: Lists

- The elements of a list don’t have to be of the same type (an element can even be another list)
- Lists are mutable (unlike strings)
- Lists operations
  - Concatenation → +
  - Repeat → [elm1, elm2] * numberOfTimes
- Slicing a list
  - myList[including:excluding]



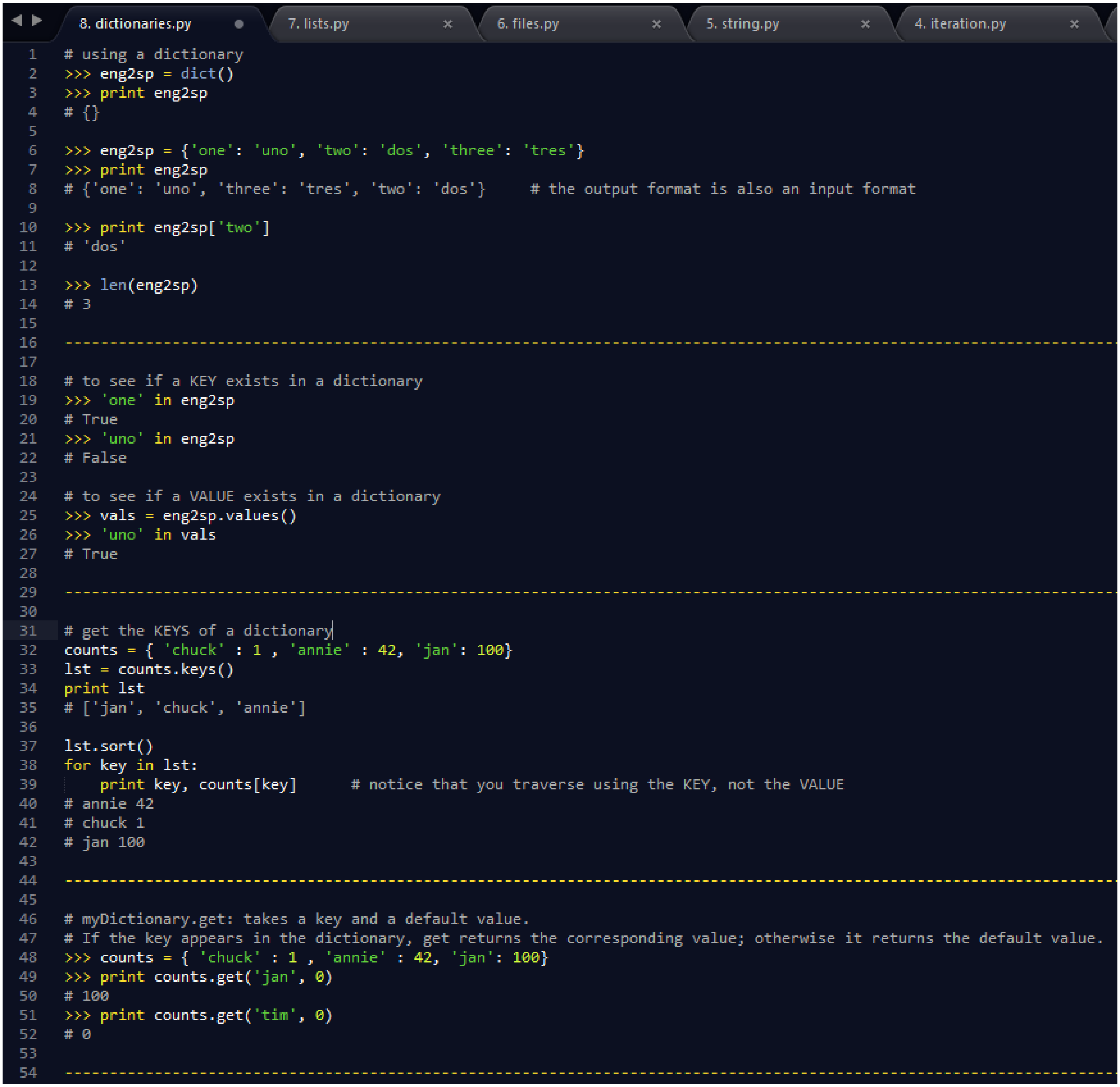
```
# using the 'in' operator
>>> cheeses = ['Cheddar', 'Edam', 'Gouda']
>>> 'Edam' in cheeses
# True
>>> 'Brie' in cheeses
# False

for cheese in cheeses:
    print cheese

-------------------------------------------------------------------------------------------------

# range([start], stop[, step])
for i in range(len(numbers)):
    numbers[i] = numbers[i] * 2

-------------------------------------------------------------------------------------------------

# append
>>> t = ['a', 'b', 'c']
>>> t.append('d')
>>> print t
# ['a', 'b', 'c', 'd']

# extend
>>> t1 = ['a', 'b', 'c']
>>> t2 = ['d', 'e']
>>> t1.extend(t2)
>>> print t1
# ['a', 'b', 'c', 'd', 'e']

# sort
>>> t = ['d', 'c', 'e', 'b', 'a']
>>> t.sort()
>>> print t
# ['a', 'b', 'c', 'd', 'e']

# pop
>>> t = ['a', 'b', 'c']
>>> x = t.pop(1)        # index of the element we want to pop (default is last element)
>>> print t
# ['a', 'c']
>>> print x
# b

# remove
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> del t[1:5]
>>> print t
# ['a', 'f']

-------------------------------------------------------------------------------------------------
```



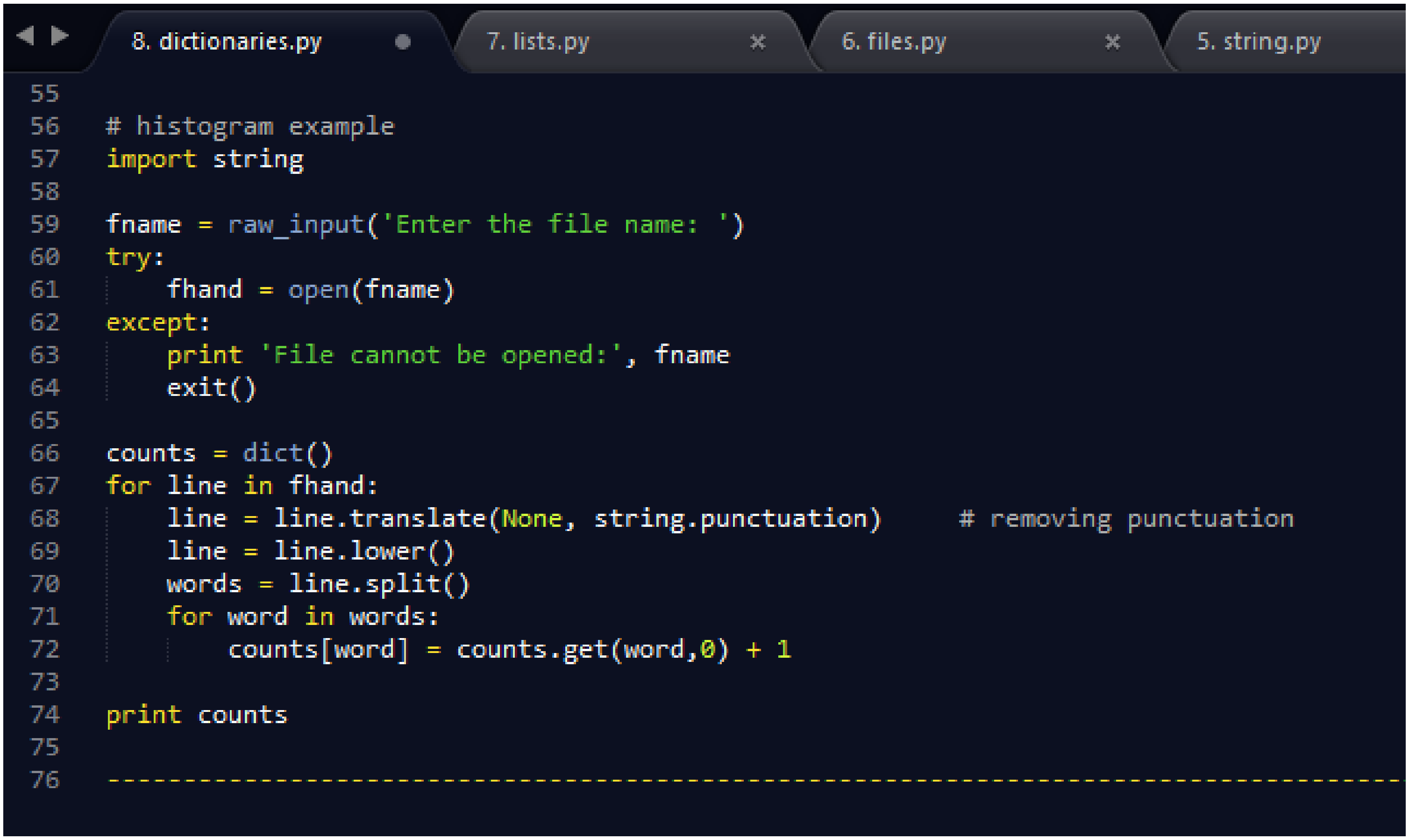
```
-----------------------------------------------------------------------------------


# list functions
>>> nums = [3, 41, 12, 9, 74, 15]
>>> print len(nums)
# 6
>>> print max(nums)
# 74
>>> print min(nums)
# 3
>>> print sum(nums)
# 154

-----------------------------------------------------------------------------------


# to convert from a string to a list of characters, you can use list
>>> s = 'spam'
>>> t = list(s)
>>> print t
# ['s', 'p', 'a', 'm']

# split
>>> s = 'pining for the fjords'
>>> t = s.split()
>>> print t
# ['pining', 'for', 'the', 'fjords']
>>> print t[2]
# the

>>> s = 'spam-spam-spam'
>>> delimiter = '-'
>>> s.split(delimiter)
# ['spam', 'spam', 'spam']

# join
>>> t = ['pining', 'for', 'the', 'fjords']
>>> delimiter = ' '
>>> delimiter.join(t)
# 'pining for the fjords'
```

Chapter 9: Dictionaries

- A dictionary is like a list, but more general. In a list, the index positions have to be integers; in a dictionary, the indices can be (almost) any type.
- The association of a key and a value is called a key-value pair or sometimes an item.
- In general, the order of items in a dictionary is unpredictable.
- THE 'in' operator uses
    - Linear search for lists
    - Uses a hash function for dictionaries
- Traversing a dictionary using 'in' gives you the KEYS, not the VALUES



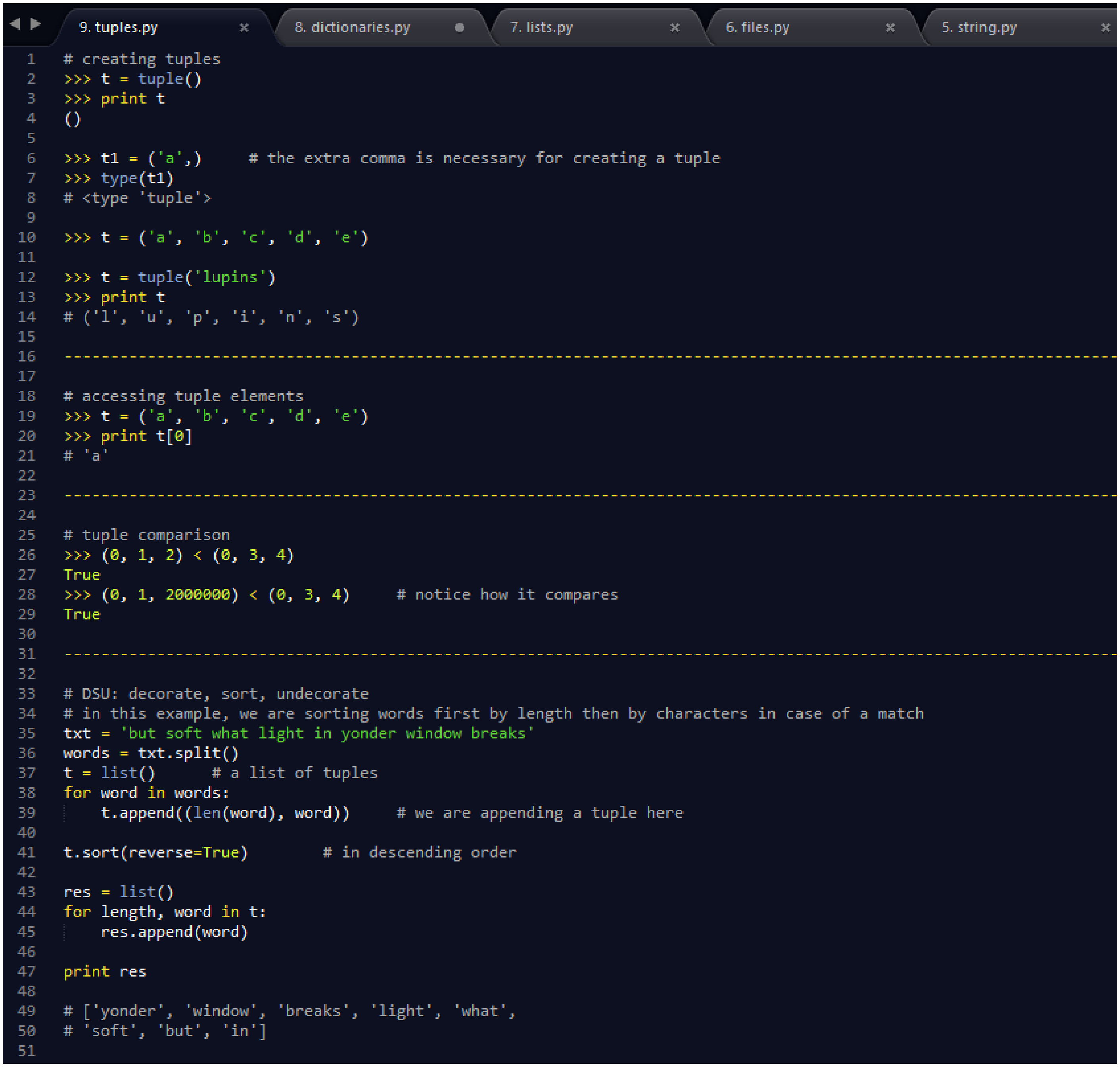
```
# using a dictionary
>>> eng2sp = dict()
>>> print eng2sp
# {}

>>> eng2sp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}
>>> print eng2sp
# {'one': 'uno', 'three': 'tres', 'two': 'dos'}     # the output format is also an input format

>>> print eng2sp['two']
# 'dos'

>>> len(eng2sp)
# 3

---------------------------------------------------------------------------------------------------------------------

# to see if a KEY exists in a dictionary
>>> 'one' in eng2sp
# True
>>> 'uno' in eng2sp
# False

# to see if a VALUE exists in a dictionary
>>> vals = eng2sp.values()
>>> 'uno' in vals
# True

---------------------------------------------------------------------------------------------------------------------

# get the KEYS of a dictionary
counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
lst = counts.keys()
print lst
# ['jan', 'chuck', 'annie']

lst.sort()
for key in lst:
    print key, counts[key]      # notice that you traverse using the KEY, not the VALUE
# annie 42
# chuck 1
# jan 100

---------------------------------------------------------------------------------------------------------------------

# myDictionary.get: 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.
>>> counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
>>> print counts.get('jan', 0)
# 100
>>> print counts.get('tim', 0)
# 0

---------------------------------------------------------------------------------------------------------------------
```



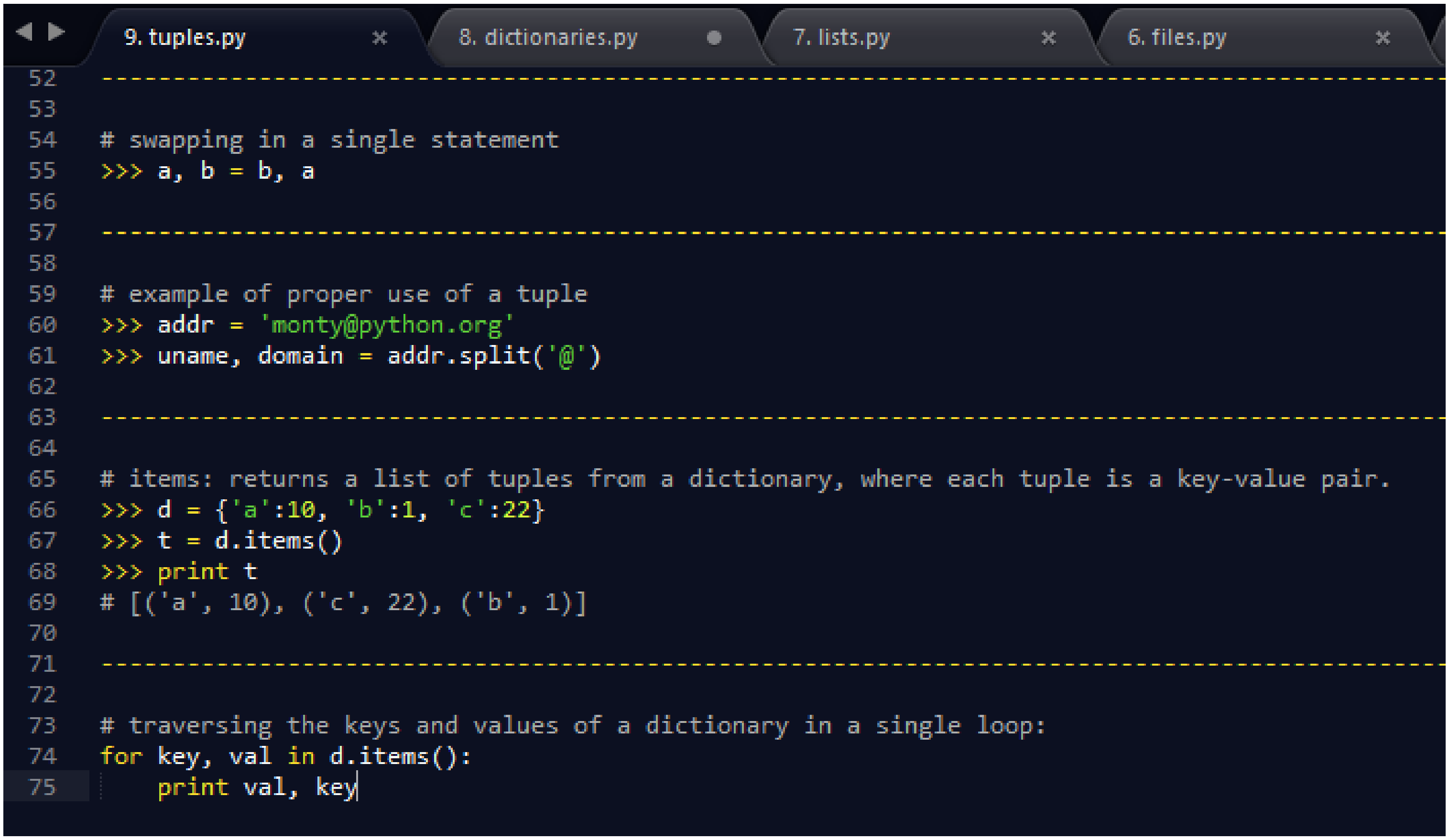
```

# histogram example
import string

fname = raw_input('Enter the file name: ')
try:
    fhand = open(fname)
except:
    print 'File cannot be opened:', fname
    exit()

counts = dict()
for line in fhand:
    line = line.translate(None, string.punctuation)     # removing punctuation
    line = line.lower()
    words = line.split()
    for word in words:
        counts[word] = counts.get(word,0) + 1

print counts


-------------------------------------------------------------------------------------
```

Chapter 10: Tuples

- Tuples are like lists, but they are
    - Immutable
    - Hashable (since they are immutable)
- Most list operations also work on tuples
- A tuple can be used as a key in a dictionary
- When to use tuples?
    - In some contexts, like a return statement, it is syntactically simpler to create a tuple than a list. In other contexts, you might prefer a list.
    - If you want to use a sequence as a dictionary key, you have to use an immutable type like a tuple or string.
    - If you are passing a sequence as an argument to a function, using tuples reduces the potential for unexpected behavior due to aliasing.

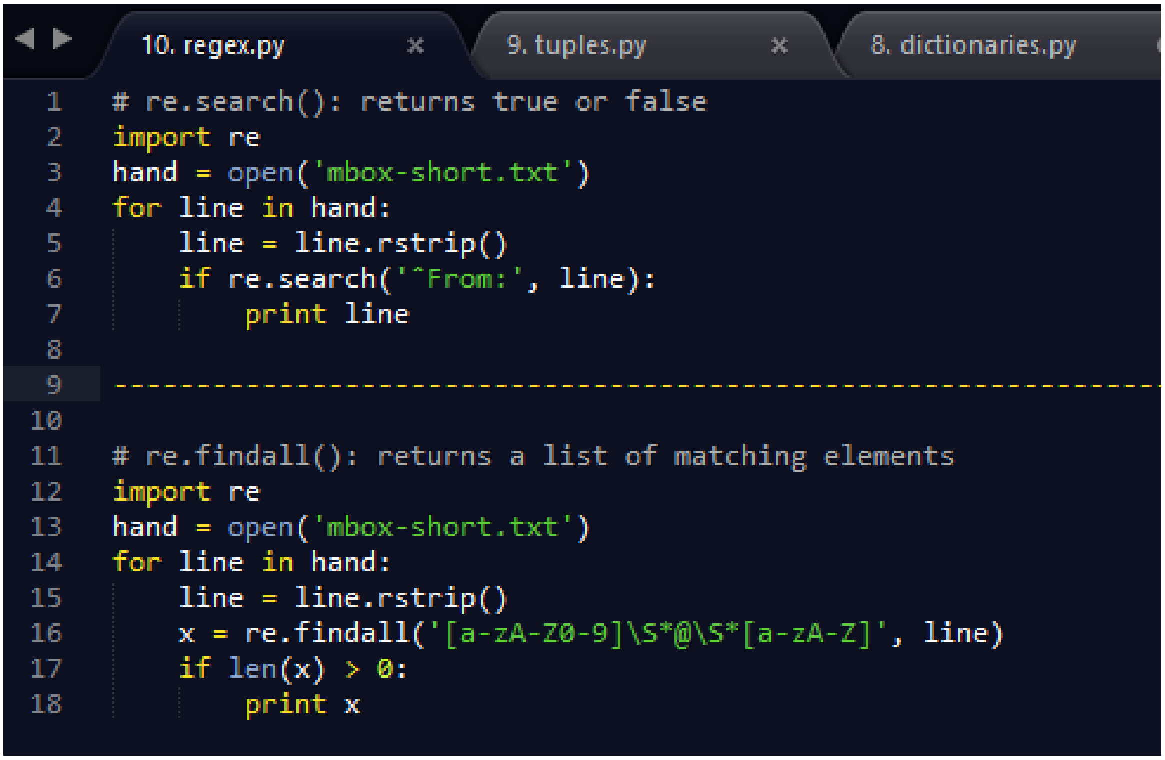


```
# creating tuples
>>> t = tuple()
>>> print t
()

>>> t1 = ('a',)     # the extra comma is necessary for creating a tuple
>>> type(t1)
# <type 'tuple'>

>>> t = ('a', 'b', 'c', 'd', 'e')

>>> t = tuple('lupins')
>>> print t
# ('l', 'u', 'p', 'i', 'n', 's')

------------------------------------------------------------------------------------------------------------------

# accessing tuple elements
>>> t = ('a', 'b', 'c', 'd', 'e')
>>> print t[0]
# 'a'

------------------------------------------------------------------------------------------------------------------


# tuple comparison
>>> (0, 1, 2) < (0, 3, 4)
True
>>> (0, 1, 2000000) < (0, 3, 4)     # notice how it compares
True


------------------------------------------------------------------------------------------------------------------

# DSU: decorate, sort, undecorate
# in this example, we are sorting words first by length then by characters in case of a match
txt = 'but soft what light in yonder window breaks'
words = txt.split()
t = list()      # a list of tuples
for word in words:
    t.append((len(word), word))     # we are appending a tuple here

t.sort(reverse=True)        # in descending order

res = list()
for length, word in t:
    res.append(word)

print res

# ['yonder', 'window', 'breaks', 'light', 'what',
# 'soft', 'but', 'in']

```



```
--------------------------------------------------------------------------------------------------------------------------------


# swapping in a single statement
>>> a, b = b, a

--------------------------------------------------------------------------------------------------------------------------------


# example of proper use of a tuple
>>> addr = 'monty@python.org'
>>> uname, domain = addr.split('@')

--------------------------------------------------------------------------------------------------------------------------------


# items: returns a list of tuples from a dictionary, where each tuple is a key-value pair.
>>> d = {'a':10, 'b':1, 'c':22}
>>> t = d.items()
>>> print t
# [('a', 10), ('c', 22), ('b', 1)]

--------------------------------------------------------------------------------------------------------------------------------


# traversing the keys and values of a dictionary in a single loop:
for key, val in d.items():
    print val, key
```

Chapter 11: Regular expressions



```python
# re.search(): returns true or false
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^From:', line):
        print line

----------------------------------------------------------------

# re.findall(): returns a list of matching elements
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall('[a-zA-Z0-9]\S*@\S*[a-zA-Z]', line)
    if len(x) > 0:
        print x
```

^ → Matches the beginning of the line.

$ → Matches the end of the line.

. → Matches any character (a wildcard).

\s → Matches a whitespace character.

\S → Matches a non-whitespace character (opposite of \s).

* → Applies to the immediately preceding character and indicates to match zero or more of the preceding character(s).

*? → Applies to the immediately preceding character and indicates to match zero or more of the preceding character(s) in “non-greedy mode”.

+ → Applies to the immediately preceding character and indicates to match one or more of the preceding character(s).

+? → Applies to the immediately preceding character and indicates to match one or more of the preceding character(s) in “non-greedy mode”.

[aeiou] → Matches a single character as long as that character is in the specified set. In this example, it would match “a”, “e”, “i”, “o”, or “u”, but no other characters.

[a-z0-9] → You can specify ranges of characters using the minus sign. This example is a single character that must be a lowercase letter or a digit.

[^A-Za-z] → When the first character in the set notation is a caret, it inverts the logic. This example matches a single character that is anything other than an uppercase or lowercase letter.

( ) → When parentheses are added to a regular expression, they are ignored for the purpose of matching, but allow you to extract a particular subset of the matched string rather than the whole string when using findall().

\b → Matches the empty string, but only at the start or end of a word.

\B → Matches the empty string, but not at the start or end of a word.

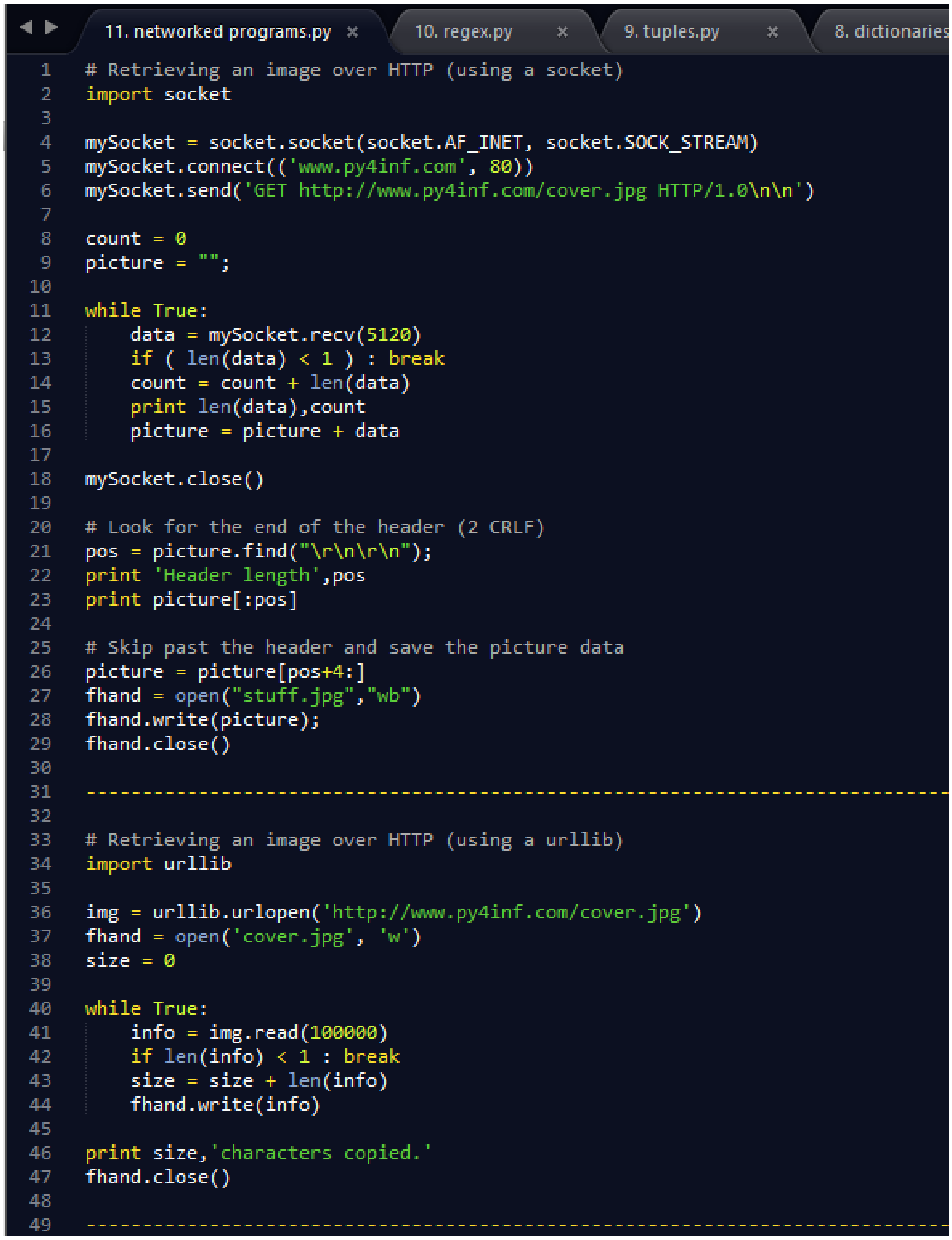
\d → Matches any decimal digit; equivalent to the set [0-9].

\D → Matches any non-digit character; equivalent to the set [ˆ0-9].

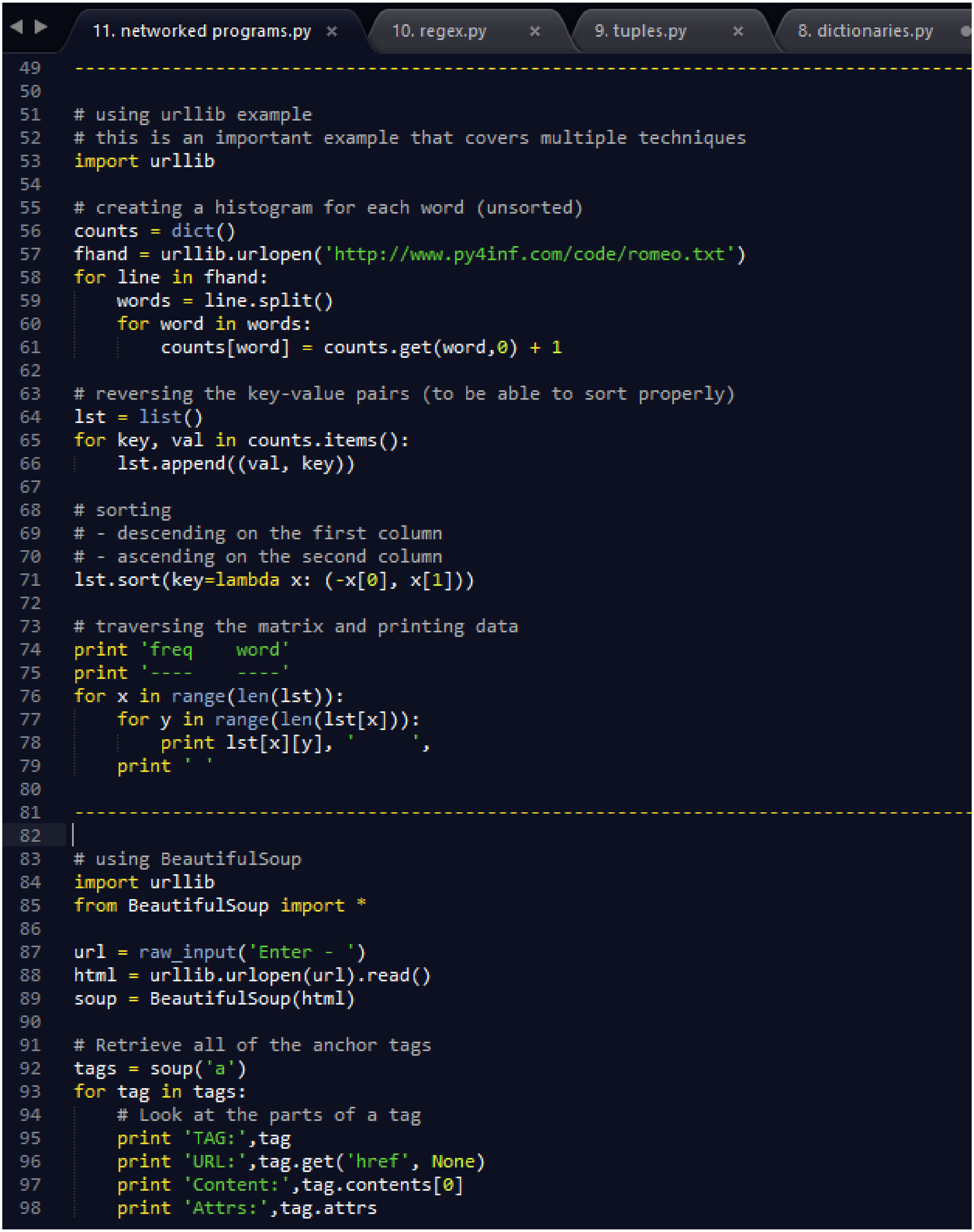
- Grep: Generalized Regular Expression Parser, does pretty much the same as re.search() (except for minor differences)

Chapter 12: Networked programs

- Protocol: a set of precise rules that determine who is to go first, what they are to do, and then what the responses are to that message, and who sends next, and so on.
- Socket: A network connection between two applications where the applications can send and receive data in either direction.
- Port: A number that generally indicates which application you are contacting when you make a socket connection to a server. As an example, web traffic usually uses port 80 while email traffic uses port 25.
- urllib: a module that provides a high-level interface for fetching data across the World Wide Web. In particular, the urlopen() function is similar to the built-in function open(), but accepts Universal Resource Locators (URLs) instead of filenames. Some restrictions apply — it can only open URLs for reading, and no seek operations are available.
- Web scraping: a computer software technique of extracting information from websites.
- Beautiful Soup: a Python library for pulling data out of HTML and XML files.
- cURL: copy url, a computer software project providing a library and command-line tool for transferring data using various protocols.



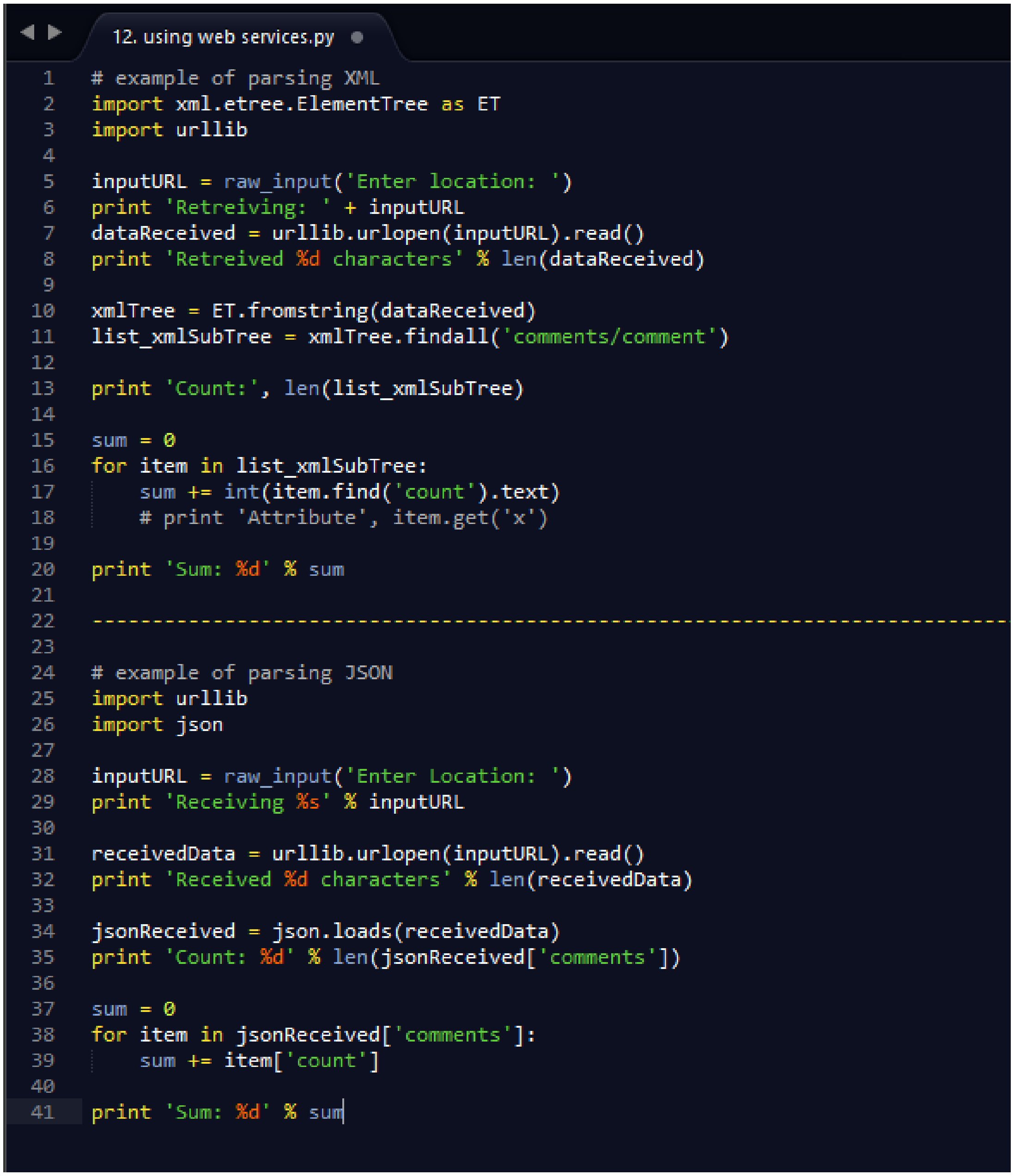
```
# Retrieving an image over HTTP (using a socket)
import socket

mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mySocket.connect(('www.py4inf.com', 80))
mySocket.send('GET http://www.py4inf.com/cover.jpg HTTP/1.0\n\n')

count = 0
picture = "";

while True:
    data = mySocket.recv(5120)
    if ( len(data) < 1 ) : break
    count = count + len(data)
    print len(data),count
    picture = picture + data

mySocket.close()

# Look for the end of the header (2 CRLF)
pos = picture.find("\r\n\r\n");
print 'Header length',pos
print picture[:pos]

# Skip past the header and save the picture data
picture = picture[pos+4:]
fhand = open("stuff.jpg","wb")
fhand.write(picture);
fhand.close()

------------------------------------------------------------------------------

# Retrieving an image over HTTP (using a urllib)
import urllib

img = urllib.urlopen('http://www.py4inf.com/cover.jpg')
fhand = open('cover.jpg', 'w')
size = 0

while True:
    info = img.read(100000)
    if len(info) < 1 : break
    size = size + len(info)
    fhand.write(info)

print size,'characters copied.'
fhand.close()

------------------------------------------------------------------------------
```



```
----------------------------------------------------------------------------------------------------------


# using urllib example
# this is an important example that covers multiple techniques
import urllib

# creating a histogram for each word (unsorted)
counts = dict()
fhand = urllib.urlopen('http://www.py4inf.com/code/romeo.txt')
for line in fhand:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word,0) + 1

# reversing the key-value pairs (to be able to sort properly)
lst = list()
for key, val in counts.items():
    lst.append((val, key))

# sorting
# - descending on the first column
# - ascending on the second column
lst.sort(key=lambda x: (-x[0], x[1]))

# traversing the matrix and printing data
print 'freq    word'
print '----    ----'
for x in range(len(lst)):
    for y in range(len(lst[x])):
        print lst[x][y], '      ',
    print ' '

----------------------------------------------------------------------------------------------------------

# using BeautifulSoup
import urllib
from BeautifulSoup import *

url = raw_input('Enter - ')
html = urllib.urlopen(url).read()
soup = BeautifulSoup(html)

# Retrieve all of the anchor tags
tags = soup('a')
for tag in tags:
    # Look at the parts of a tag
    print 'TAG:',tag
    print 'URL:',tag.get('href', None)
    print 'Content:',tag.contents[0]
    print 'Attrs:',tag.attrs
```

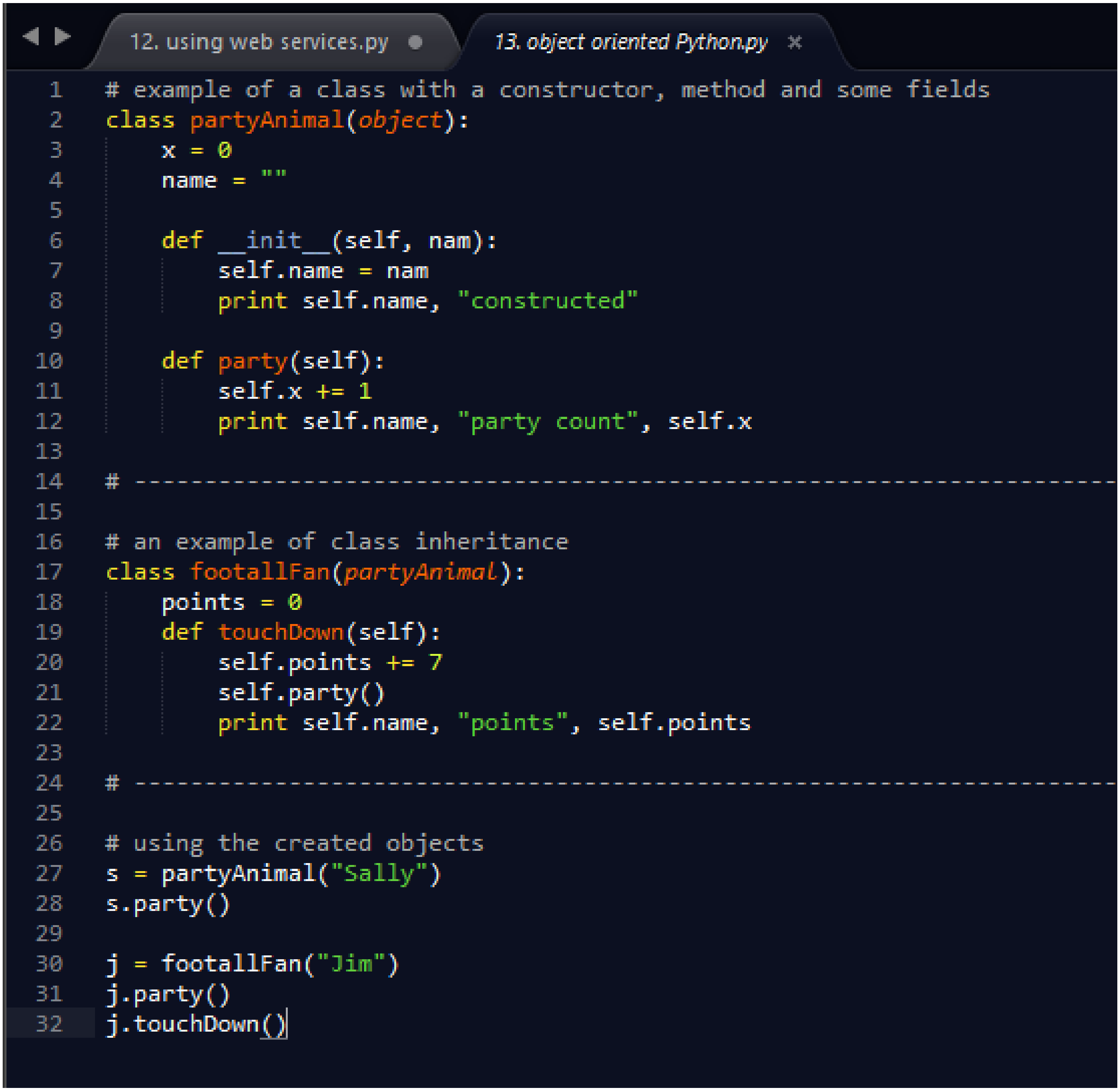
Chapter 13: Using web services

- To exchange data across the web, you can use
  - XML
    - more suited for document style data (e.g. docx)
  - JSON
    - more suited for simple data structures
    - maps directly to a Python dictionary and a list objects
- SOA: Service-Oriented Architecture. When an application is made of components connected across a network.
  - When an application makes a set of services in its API available over the web, we call these web services.
- OAuth: an open standard for authorization, commonly used as a way for Internet users to log in to third party websites using their Microsoft, Google, Facebook, Twitter, One Network etc. accounts without exposing their password.
- When using a web service, you need to read the documentation provided and abide by the rules of usage mentioned (especially regarding how many times you can use the service in a given day)



```python
# example of parsing XML
import xml.etree.ElementTree as ET
import urllib

inputURL = raw_input('Enter location: ')
print 'Retreiving: ' + inputURL
dataReceived = urllib.urlopen(inputURL).read()
print 'Retrieved %d characters' % len(dataReceived)

xmlTree = ET.fromstring(dataReceived)
list_xmlSubTree = xmlTree.findall('comments/comment')

print 'Count:', len(list_xmlSubTree)

sum = 0
for item in list_xmlSubTree:
    sum += int(item.find('count').text)
    # print 'Attribute', item.get('x')

print 'Sum: %d' % sum

----------------------------------------------------------------------------

# example of parsing JSON
import urllib
import json

inputURL = raw_input('Enter Location: ')
print 'Receiving %s' % inputURL

receivedData = urllib.urlopen(inputURL).read()
print 'Received %d characters' % len(receivedData)

jsonReceived = json.loads(receivedData)
print 'Count: %d' % len(jsonReceived['comments'])

sum = 0
for item in jsonReceived['comments']:
    sum += item['count']

print 'Sum: %d' % sum
```

Objected oriented Python



```
# example of a class with a constructor, method and some fields
class partyAnimal(object):
    x = 0
    name = ""

    def __init__(self, nam):
        self.name = nam
        print self.name, "constructed"

    def party(self):
        self.x += 1
        print self.name, "party count", self.x

# ----------------------------------------------------------------------

# an example of class inheritance
class footallFan(partyAnimal):
    points = 0
    def touchDown(self):
        self.points += 7
        self.party()
        print self.name, "points", self.points

# ----------------------------------------------------------------------

# using the created objects
s = partyAnimal("Sally")
s.party()

j = footallFan("Jim")
j.party()
j.touchDown()
```

Chapter 14: Using databases and Structured Query Language (SQL)

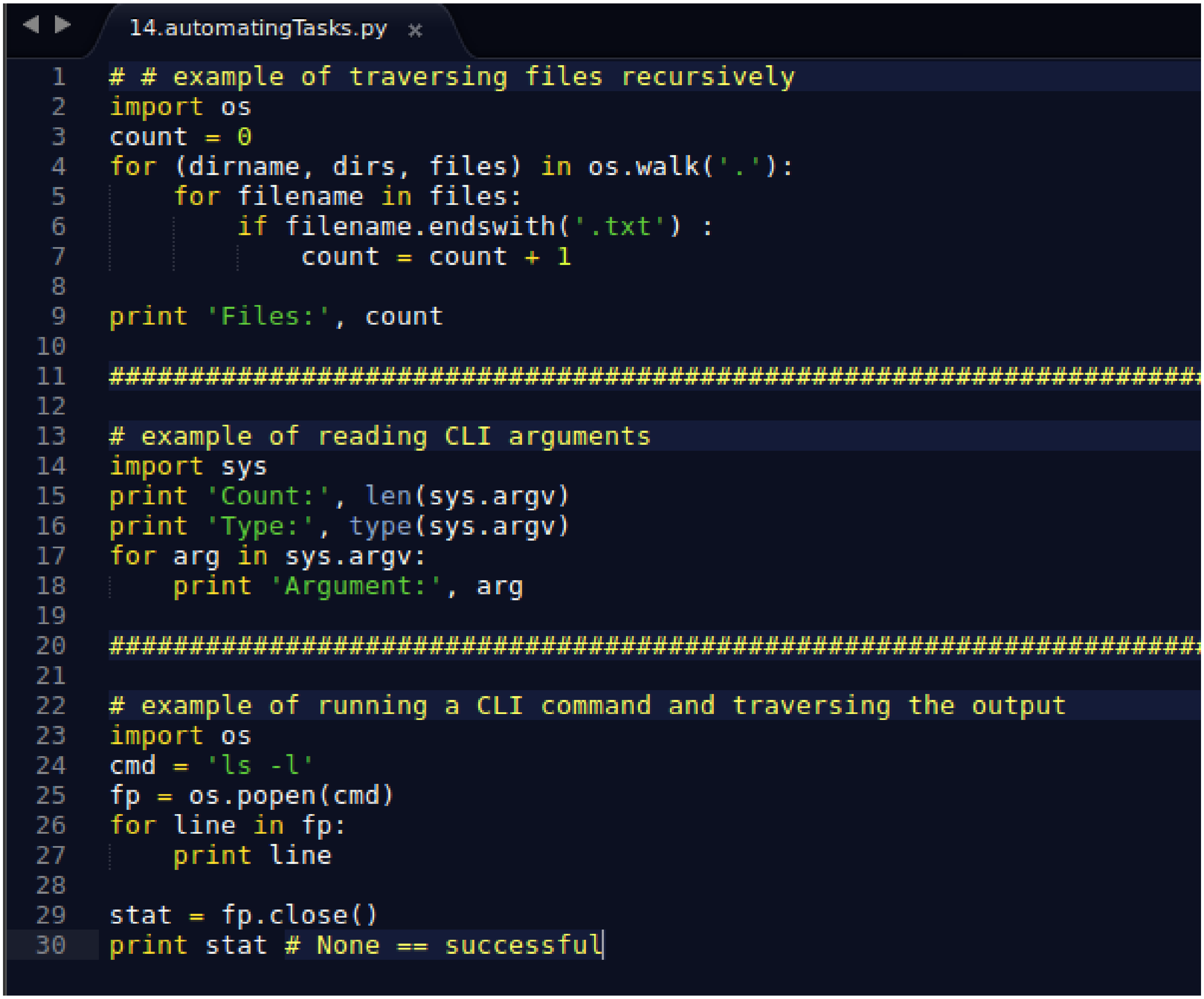
- A database is a file that is organized for storing data.
- Database primary constructs
  - Table → relation
  - Row → tuple
  - Column → attribute
- Database basic operations → CRUD
  - Create
  - Read
  - Update
  - Delete
- Unlike in Python, in a SQL WHERE clause we use a single equal sign to indicate a test for equality rather than a double equal sign.
- A DBMS maintains its performance by building indexes as data is added to the database to allow the computer to jump quickly to a particular entry.
- SQLite: a relational database management system contained in a C programming library. In contrast to many other database management systems, SQLite is not a client–server database engine. Rather, it is embedded into the end program.
- A cursor is like a file handle (or a socket) that we can use to perform operations on the data stored in the database. Calling "cursor()" is very similar conceptually to calling "open()" when dealing with text files.
- Always use the provided database module method to insert values into your query. This is important to avoid SQL injections.
- Data mining technologies (for future reference):
  - Hadoop
  - Spark apache
  - Aws redshift
  - Pentaho
- The act of deciding how to break up your application data into multiple tables and establishing the relationships between the tables is called "data modeling". The design document that shows the tables and their relationships is called a "data model".
- Types of keys:
  - Logical key → a key that the "real world" might use to look up a row
  - Primary key → a number that is assigned automatically by the database.
  - Foreign key → A foreign key is usually a number that points to the primary key of an associated row in a different table
- You should never put the same string data in the database more than once. If you need the data more than once, create a numeric key for the data and reference the actual data using this key.
- Use a naming convention of always calling the primary key field name "id" and appending the suffix "_id" to any field name that is a foreign key.
- Don't create a many-to-many relation, always break it into a one to many relation. This is done by creating a new relation in the middle that has the primary key of both as foreign keys (this combination will be the primary key for the new relation – called a composite key)

- The situations where a database can be useful are:
  - When your application needs to make small many random updates within a large data set
  - When your data is so large it cannot fit in a dictionary and you need to look up information repeatedly
  - When you have a long-running process that you want to be able to stop and restart and retain the data from one run to the next.
- General steps for using SQLite in Python
  - MyConnection = Sqlite3.connect('databaseName.sqlite') → note that a new DB will be created if the database does not exist
  - myCursor = myConnection.cursor() → this is like a file handle or a socket
  - myCursor.execute(SQL_Statement)
  - myConnection.commit() → this will write the changes made to disk for everyone to see
  - myConnection.close()

➔ for a code example, refer to "14. databases.py"

Chapter 16: Automating common tasks on your computer

- Relative paths start from the current directory
- Absolute paths start from the topmost directory in the file system.
- The OS module in Python provides a way of using operating system dependent functionality.
  - os.getcwd() → Returns the current working directory
  - os.path.abspath(fileName.txt') → find the absolute path to a file
  - os.path.exists('memo.txt') → checks whether a file or directory exists
  - os.path.isdir('memo.txt') →checks whether it's a directory
  - os.path.isfile() → checks if the argument is a file
  - os.listdir(cwd) → returns a list of the files (and other directories) in the given directory
  - os.remove(thefile) → delete a file
  - os.walk('.') → will "walk" through all of the directories and subdirectories recursively. The string "." indicates to start in the current directory and walk downward. As it encounters each directory, we get three values in a tuple in the body of the for loop. The first value is the current directory name, the second value is the list of subdirectories in the current directory, and the third value is a list of files in the current directory



```
# # example of traversing files recursively
import os
count = 0
for (dirname, dirs, files) in os.walk('.'):
    for filename in files:
        if filename.endswith('.txt') :
            count = count + 1

print 'Files:', count

####################################################################

# example of reading CLI arguments
import sys
print 'Count:', len(sys.argv)
print 'Type:', type(sys.argv)
for arg in sys.argv:
    print 'Argument:', arg

####################################################################

# example of running a CLI command and traversing the output
import os
cmd = 'ls -l'
fp = os.popen(cmd)
for line in fp:
    print line

stat = fp.close()
print stat # None == successful
```