Lecture 4-3

Working with files

Week 4 Friday

Miles Chen, PhD

Adapted from Chapter 14 of Think Python by Allen B Downey

Persistence

Most of the code we have run so far are transient. They will run during your session and after producing some output, their data disappears.
Other programs need to be **persistent**, meaning that they need to store (at least some of) their data.
This chapter will focus on reading and writing files to safe data.

Opening a file

You can open a file with `open()`
You can write a to a file using `'w'` as a second parameter:
`open('filename.txt', 'w')`
If the file exists, running this command will replace the file. If the file does not exist, it will be created.

In [1]:

```
fout = open('output.txt', 'w')
```

The function `open()` returns a file object that allows you to write to the file.

In [2]:

```
line1 = "This is the first line of text.\n"
fout.write(line1)
```

Out[2]:

```
32
```

The method returns the number of characters that were written.

If you call write again, it adds the new data to the end of the file.

```
In [3]:
```

```
line2 = "This will be the second line of text.\n"
fout.write(line2)
```

```
Out[3]:

38
```

When you are done writing, you should close the file.

```
In [4]:
```

```
fout.close()
```

Closing the file frees up the system resources dedicated to tracking that file. If you forget to close the file and Python sees no object names bound to the file object, Python's garbage collector will automatically close the connection.

with

You can use the `with` keyword to have operations run with a particular file. When the operations finish, Python will automatically close the file.

In [5]:

```
# opening a file and writing lines to it:
with open('output.txt', 'w') as file:
    file.write(line1)
    file.write(line2)
```

In [6]:

```
# opening the file and printing each line
with open('output.txt', 'r') as file:
    for i in file:
        print(i)
```

```
This is the first line of text.

This will be the second line of text.
```

The Format Operator

When using `write`, the argument has to be a string.
The easiest way to convert non-strings to strings is with `str()`. If you don't convert the value to a string, you can run into problems.

In [7]:

```
x = 42.3
y = x + " is my favorite number"
```

```
---------------------------------------------------------------------------
TypeError
 Traceback (most recent call last)
<ipython-input-7-bd11443d914f> in <module>
      1 x = 42.3
----> 2 y = x + " is my favorite number"

TypeError: unsupported operand type(s) for
+: 'float' and 'str'
```

In [8]:

```
x = 42.3
y = str(x) + " is my favorite number"
print(y)
```

```
42.3 is my favorite number
```

Another option is to use the format operator `%`. This is the same symbol for modulo division, but can be applied to format strings.
Python has **format sequences** which can be used to specify the format of the string.
See: **https://docs.python.org/3/library/stdtypes.html#printf-style-string-formatting**
For example, `'%d'` will tell Python to format the value as an integer.
`'%g'` will tell Python to format the value as a floating point number. It switches to exponential format if the exponent is less than -4.
`'%f'` will tell Python to format the value as a floating point number.
The format operator can be placed inline into any string and Python will replace the format sequence with the appropriate string.

In [9]:

```
x = 12.34
y = '%d is my favorite number' % x
print(y)
```

```
12 is my favorite number
```

In [10]:

```
x = 12.34
y = '%g is my favorite number' % x
print(y)
```

12.34 is my favorite number

In [11]:

```
x = 0.0001234
y = '%g is my favorite number' % x
print(y)
```

```
0.0001234 is my favorite number
```

In [12]:

```
x = 0.00001234
y = '%g is my favorite number' % x
print(y)
```

```
1.234e-05 is my favorite number
```

In [13]:

```
# default is 6 decimals of precision
x = 0.00001234
y = '%f is my favorite number' % x
print(y)
```

```
0.000012 is my favorite number
```

In [14]:

```
# you can specify more decimals of precision after the dot
x = 0.00001234
y = '%.10f is my favorite number' % x
print(y)
```

```
0.0000123400 is my favorite number
```

The format operator can also be used with multiple placeholders and a tuple of values.

In [15]:

```python
x = 12.345
age = 85
y = 'My age is %d, and %g is my favorite number' % (age, x)
print(y)
```

```
My age is 85, and 12.345 is my favorite nu
mber
```

In [16]:

```python
# alternative method using + operator
x = 12.345
age = 85
y = 'My age is ' + str(age) + ', and ' + str(x) + ' is my favorite number'
print(y)
```

```
My age is 85, and 12.345 is my favorite nu
mber
```

In [17]:

```python
y = 'My name is %s, I am %d years old, and %g is my favorite number' % ("Joe Bruin", 100, 83)
print(y)
```

My name is Joe Bruin, I am 100 years old,
and 83 is my favorite number

Filenames and Paths

Python's `os` module allows you to work with your computer's file system in case you need to work with directories.

```
In [18]:
```

```
import os
```

```
In [19]:
```

```
# Use os.getcwd() to get the current working directory
cwd = os.getcwd()
cwd
```

```
Out[19]:
```

```
'C:\\Users\\miles\\onedrive\\Teaching\\21
\\2021-sp-stats21'
```

When you provide a filename to Python, it is generally a relative path because it relates to the current working directory.
You can ask for absolute paths with `os.path.abspath()`

```
In [20]:
```

```
os.path.abspath('output.txt')
```

```
Out[20]:

'C:\\Users\\miles\\onedrive\\Teaching\\21
\\2021-sp-stats21\\output.txt'
```

more os functions

You can read more about Python's OS functions at:
https://docs.python.org/3/library/os.html

```
In [21]:
```

```
# check if a file exists
os.path.exists('memo.txt')
```

```
Out[21]:
```

```
False
```

```
In [22]:
```

```
os.path.exists('output.txt')
```

```
Out[22]:
```

```
True
```

```
In [23]:
```

```
# check if something is a directory
os.path.isdir('output.txt')
```

```
Out[23]:
```

```
False
```

In [24]:

```
os.path.isdir('.git')
```

Out[24]:

```
True
```

In [25]:

```python
# list the contents of the current working directory
os.listdir(cwd)
```

Out[25]:

```
['.git',
 '.gitignore',
 '.ipynb_checkpoints',
 '1-1_Lecture_Stats_21.pdf',
 '1-2_Lecture_Stats_21.pdf',
 '1-3_Lecture_Stats_21.ipynb',
 '1-3_Lecture_Stats_21.pdf',
 '1-3_Lecture_Stats_21.slides.html',
 '2-1_Lecture_Stats_21.ipynb',
 '2-1_Lecture_Stats_21.pdf',
 '2-1_Lecture_Stats_21.slides.html',
 '2-2_Lecture_Stats_21.ipynb',
 '2-2_Lecture_Stats_21.pdf',
 '2-2_Lecture_Stats_21.slides.html',
 '2-3_Lecture_Stats_21.ipynb',
 '2-3_Lecture_Stats_21.pdf',
 '2-3_Lecture_Stats_21.slides.html',
 '3-1_Lecture_Stats_21.ipynb',
 '3-1_Lecture_Stats_21.pdf',
 '3-1_Lecture_Stats_21.slides.html',
 '3-2_Lecture_Stats_21.ipynb',
 '3-2_Lecture_Stats_21.pdf',
 '3-2_Lecture_Stats_21.slides.html',
 '3-3_Lecture_Stats_21.ipynb',
 '3-3_Lecture_Stats_21.pdf',
 '3-3_Lecture_Stats_21.slides.html',
 '4-1_Lecture_Stats_21.ipynb',
 '4-1_Lecture_Stats_21.pdf',
 '4-1_Lecture_Stats_21.slides.html',
 '4-2_Lecture_Stats_21.ipynb',
 '4-2_Lecture_Stats_21.pdf',
 '4-2_Lecture_Stats_21.slides.html',
 '4-3_Lecture_Stats_21.ipynb',
 '4-3_Lecture_Stats_21.slides.html',
 'captions.bak',
    'captions.dat',
    'captions.dir',
    'objects.pkl',
    'output.txt',
    'README.md',
    'script01.py',
    'script02.py',
    '__pycache__']
```

Catching Exceptions

When dealing with files, and also just in programming in general, a lot of things can go wrong and Python will throw an exception.
You can often catch many of the exceptions with `try ... except` which works in a manner similar to `if ... else`

In [26]:

```
fin = open('bad_file')
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-26-a7d7d7ad396b> in <module>
----> 1 fin = open('bad_file')
```

```
FileNotFoundError: [Errno 2] No such file
 or directory: 'bad_file'
```

```
In [27]:
```

```python
try:
    fin = open('bad_file')
except:
    print('Something went wrong.')
```

```
Something went wrong.
```

Python first tries the commands in the `try:` block. If everything goes well, it skips the commands in the `except:` block. If it encounters an exception in the `try` block, it immediately exits the block and executes the code in the `except` block.

Python dbm databases

A Python dbm database is like a Python dictionary but saved to a file on the harddrive.

In [28]:

```
# You first have to import the dbm library
import dbm
```

`dbm.open()` will open the database connection
The 'c' option will create it if it doesn't yet exist. It will not overwrite an existing file

In [29]:

```
db = dbm.open('captions', 'c')
```

Add keys and values much like a dictionary.

In [30]:

```
db['cleese.png'] = 'Photo of John Cleese'
```

The key here is the string 'cleese.png'. It is not a file, just the string of a filename (that may or may not exist). The value is another string.

In [31]:

```
# retrieve values like a dictionary
db['cleese.png']
```

Out[31]:

```
b'Photo of John Cleese'
```

In [32]:

```python
# I add another key-value pair.
db['jones.png'] = 'Photo of Terry Jones'
# if you need to, you can also delete an entry using del(db['jones.png'])
```

In [33]:

```python
# you can iterate through a database
for key in db:
    print(key, db[key])
```

```
b'cleese.png' b'Photo of John Cleese'
b'jones.png' b'Photo of Terry Jones'
b'list' b'\x80\x03]q\x00(K\x01K\x02K\x03
e.'
```

In [34]:

```python
# you should close when done
db.close()
```

In [35]:

```python
# creating the captions dbm will add a few files to your working directory:
for i in os.listdir(cwd):
    if os.path.isfile(i) and 'captions' in i:
        print(i)
```

```
captions.bak
captions.dat
captions.dir
```

Python Pickles

A major limitation with dbm is that it can only store strings or byte objects. If you want to store other objects you created in Python, you'll want to `pickle` them. This converts the Python object into a string that can then be written to the dbm.

```
In [36]:
```

```
t = [1 ,2, 3]
type(t)
```

```
Out[36]:
```

```
list
```

```
In [37]:
```

```
db = dbm.open('captions', 'c')
```

```
In [38]:
```

```
# if I try to save the list t to the dbm, it causes an error.
db['list'] = t
```

```
---------------------------------------------------------------------------
TypeError
 Traceback (most recent call last)
<ipython-input-38-f05b3aca1a75> in <module
>
      1 # if I try to save the list t to t
he dbm, it causes an error.
----> 2 db['list'] = t

~\anaconda3\lib\dbm\dumb.py in __setitem__
(self, key, val)
    202                 val = val.encode('utf-
8')
    203             elif not isinstance(val, (
bytes, bytearray)):
--> 204                 raise TypeError("value
s must be bytes or strings")
    205             self._verify_open()
    206             self._modified = True
```

```
TypeError: values must be bytes or strings
```

In [39]:

```
import pickle
```

In [40]:

```
# pickle.dumps converts the python object to a string
s = pickle.dumps(t)
```

In [41]:

```
# the string representation of the list t
# This is not meant to be human-readable. It is simply a format that can be stored in the dbm.
s
```

Out[41]:

```
b'\x80\x03]q\x00(K\x01K\x02K\x03e.'
```

In [42]:

```
db['list'] = s
```

In [43]:

```
db.close()
```

Now that we have saved the "pickled" list to the dbm, let's see if we can retreive it.

In [44]:

```
# open the dbm
db = dbm.open('captions', 'c')
```

In [45]:

```
# retrieve the string associated with the key 'list'
db['list']
```

Out[45]:

```
b'\x80\x03]q\x00(K\x01K\x02K\x03e.'
```

In [46]:

```
# to conver the string back to a python object, you can use pickle.loads()
pickle.loads(db['list'])
```

Out[46]:

```
[1, 2, 3]
```

You can pickle objects directly to a pickle file.
You can pickle multiple objects by grouping them together in a list or tuple.

In [47]:

```
obj0 = ['file1', 'something else']
obj1 = [1, 2, 3]
obj2 = "hello world!"

with open('objects.pkl', 'wb') as file:
    pickle.dump([obj0, obj1, obj2], file)
```

Loading the pickle

```
In [48]:
with open('objects.pkl', "rb") as file:
    x1, x2, x3 = pickle.load(file)
```

```
In [49]:
x1
```

```
Out[49]:
['file1', 'something else']
```

```
In [50]:
x2
```

```
Out[50]:
[1, 2, 3]
```

```
In [51]:
x3
```

```
Out[51]:
'hello world!'
```

Read more about pickling:
https://docs.python.org/3/library/pickle.html#what-can-be-pickled-and-unpickled