23 File Input/Output

"Save in file, or perish..."

Contents

I/O System

- Expectations from an I/O System:
  - It should allow us to communicate with multiple sources and destinations.
    Ex. Sources - Keyboard, File, Network
    Ex. Destinations - Screen, File, Network
  - It should allow us to input/output varied entities.
    Ex. Numbers, Strings, Lists, Tuples, Sets, Dictionaries, etc.
  - It should allow us to communicate in multiple ways.
    Ex. Sequential access, Random access
  - It should allow us to deal with underlying file system.
    Ex. Create, Modify, Rename, Delete files and directories
- Types of data used for I/O:

  Text - '485000' as a sequence of Unicode characters.
  Binary - 485000 as sequence of bytes of its binary equivalent.
- File Types:

  All program files are text files.
  All image, music, video, executable files are binary files.

File I/O

- Sequence of operations in file I/O:
  - Open a file
  - Read/Write data to it
  - Close the file
- Given below is a program that implements this sequence of file I/O operations:

```
# write/read text data
msg1 = 'Pay taxes with a smile...\n'
msg2 = 'I tried, but they wanted money!\n'
msg3 = 'Don\'t feel bad...\n'
msg4 = 'It is alright to have no talent!\n'
f = open('messages', 'w')
f.write(msg1)
f.write(msg2)
```

```
f.write(msg3)
f.write(msg4)
f.close( )
f = open('messages', 'r')
data = f.read( )
print(data)
f.close( )
```

On executing this program, we get the following output:

```
Pay taxes with a smile...
I tried, but they wanted money!
Don't feel bad...
It is alright to have no talent!
```

- Opening a file brings its contents to a buffer in memory. While performing read/write operations, data is read from or written to buffer.

```
f = open(filename, 'r')      # opens file for reading
f = open(filename, 'w')      # opens file for writing
f.close( )                   # closes the file by vacating the buffer
```

Once file is closed read/write operation on it are not feasible.

- **f.write(msg1)** writes **msg1** string to the file.
- **data = f.read( )** reads all the lines present in the file represented by object **f** into **data**.

Read / Write Operations

- There are two functions for writing data to a file:

```
msg = 'Bad officials are elected by good citizens who do not vote.\n'
msgs = ['Humpty\n', 'Dumpty\n', 'Sat\n', 'On\n', 'a\n', 'wall\n']
f.write(msg)
f.writelines(msgs)
```

- To write objects other than strings, we need to convert them to strings before writing:

```
tpl = ('Ajay', 23, 15000)
```

```
lst = {23, 45, 56, 78, 90}
d = {'Name' : 'Dilip', 'Age' : 25}
f.write(str(tpl))
f.write(str(lst))
f.write(str(d))
```

- There are three functions for reading data from a file represented by file object **f**.

```
data = f.read( )        # reads entire file contents and returns as string
data = f.read(n)        # reads n characters, and returns as string
data = f.readline( )    # reads a line, and returns as string
```

If end of file is reached **f.read( )** returns an empty string.

- There are two ways to read a file line-by-line till end of file:

```
# first way
while True :
    data = f.readline( )
    if data == '' :
        break
    print(data, end ='')

# second way
for data in f :
    print(data, end ='')
```

- To read all the lines in a file and form a **list** of lines:

```
data = f.readlines( )
```

File Opening Modes

- There are multiple file-opening modes available:

'r' - Opens file for reading in text mode.
'w' - Opens file for writing in text mode.
'a' - Opens file for appending in text mode.
'r+' - Opens file for reading and writing in text mode.
'w+' - Opens file for writing and reading in text mode.
'a+' - Opens file for appending and reading in text mode.

'rb' - Opens file for reading in binary mode.
'wb' - Opens file for writing in binary mode.
'ab' - Opens file for appending in binary mode.
'rb+' - Opens file for reading and writing in binary mode.
'wb+' - Opens file for writing and reading in binary mode.
'ab+' - Opens file for appending and reading in binary mode.

If mode argument is not mentioned while opening a file, then 'r' is assumed.

- While opening a file for writing, if the file already exists, it is overwritten.
- If file is opened for writing in binary mode then a bytes-like object should be passed to **write( )** as shown below:

```
f = open('a.dat', 'wb+')
d = b'\xee\x86\xaa'      # series of 3 bytes, \x indicates hexadecimal
f.write(d)
```

with Keyword

- It is a good idea to close a file once its usage is over, as it will free up system resources.
- If we don't close a file, when the file object is destroyed file will be closed for us by Python's garbage collector program.
- If we use **with** keyword while opening the file, the file gets closed as soon as its usage is over.

```
with open('messages', 'r') as f :
    data = f.read( )
```

- **with** ensures that the file is closed even if an exception occurs while processing it.

Moving within a File

- When we are reading a file or writing a file, the next read or write operation is performed from the next character/byte as compared to the previous read/write operation.
- Thus if we read the first character from a file using **f.read(1)**, next call to **f.read(1)** will automatically read the second character in the file.

- At times we may wish to move to desired position in a file before reading/writing. This can be done using **f.seek( )** method.
- General form of **seek( )** is given below:

  f.seek(offset, reference)

  reference can take values 0, 1, 2 standing for beginning of file, current position in file and end of file respectively.
- For file opened in text mode, reference values 0 and 2 alone can be used. Also, using 2, we can only move to end of file.

```
f.seek(512, 0)   # moves to position 512 from beginning of file
f.seek(0, 2)     # moves to end of file
```

- For file opened in binary mode, reference values 0, 1, 2 can be used.

```
f.seek(0)        # moves to beginning of file
f.seek(12, 0)    # moves to position 12 from beginning of file
f.seek(-15, 2)   # moves 15 positions to left from end of file
f.seek(6, 1)     # moves 6 positions to right from current position
f.seek(-10, 1)   # moves 10 positions to left from current position
```

Serialization and Deserialization

- Compared to strings, reading/writing numbers from/to a file is tedious. This is because **write( )** writes a string to a file and **read( )** returns a string read from a file. So we need to do conversions while reading/writing, as shown in the following program:

```
f = open('numberstxt', 'w+')
f.write(str(233)+'\n')
f.write(str(13.45))
f.seek(0)
a = int(f.readline( ))
b = float(f.readline( ))
print(a + a)
print(b + b)
```

- If we are to read/write more complicated data in the form of tuple, dictionaries, etc. from/to file using the above method, it will become more difficult. In such cases a module called **json** should be used.

- **json** module converts Python data into appropriate JSON types before writing data to a file. Likewise, it converts JSON types read from a file into Python data. The first process is called **serialization** and the second is called **deserialization**.

```
# serialize/deserialize a list
import json
f = open('sampledata', 'w+')
lst = [10, 20, 30, 40, 50, 60, 70, 80, 90]
json.dump(lst, f)
f.seek(0)
inlst = json.load(f)
print(inlst)
f.close( )

# serialize/deserialize a tuple
import json
f = open('sampledata', 'w+')
tpl = ('Ajay', 23, 2455.55)
json.dump(tpl, f)
f.seek(0)
intpl = json.load(f)
print(tuple(intpl))
f.close( )

# serialize/deserialize a dictionary
import json
f = open('sampledata', 'w+')
dct = { 'Anil' : 24, 'Ajay' : 23, 'Nisha' : 22}
json.dump(dct, f)
f.seek(0)
indct = json.load(f)
print(indct)
f.close( )
```

- Serialization of a Python type to JSON data is done using a function **dump( )**. It writes the serialized data to a file.

- Deserialization of a JSON type to a Python type is done using a function **load( )**. It reads the data from a file, does the conversion and returns the converted data.

- While deserializing a tuple, **load( )** returns a list and not a tuple. So we need to convert the list to a tuple using **tuple( )** conversion function.
- Instead of writing JSON data to a file, we can write it to a string, and read it back from a string as shown below:

```
import json
lst = [10, 20, 30, 40, 50, 60, 70, 80, 90]
tpl = ('Ajay', 23, 2455.55)
dct = { 'Anil' : 24, 'Ajay' : 23, 'Nisha' : 22}

str1 = json.dumps(lst)
str2 = json.dumps(tpl)
str3 = json.dumps(dct)
l = json.loads(str1)
t = tuple(json.loads(str2))
d = json.loads(str3)
print(l)
print(t)
print(d)
```

- It is possible to serialize/deserialize nested lists and directories as shown below:

```
# serialize/deserialize a dictionary
import json
lofl = [10, [20, 30, 40], [ 50, 60, 70], 80, 90]
f = open('data', 'w+')
json.dump(lofl, f)
f.seek(0)
inlofl = json.load(f)
print(inlofl)
f.close( )

# serialize/deserialize a dictionary
import json
contacts = { 'Anil': { 'DOB' : '17/11/98', 'Favorite' : 'Igloo' },
             'Amol': { 'DOB' : '14/10/99', 'Favorite' : 'Tundra' },
             'Ravi': { 'DOB' : '19/11/97', 'Favorite' : 'Artic' } }
f = open('data', 'w+')
json.dump(contacts, f)
f.seek(0)
```

```
incontacts = json.load(f)
print(incontacts)
f.close( )
```

Serialization of User-defined Types

- Standard Python types can be easily converted to JSON and vice-cersa. However, if we attempt to serialize a user-defined **Complex** type to JSON we get following error:

  TypeError: Object of type 'Complex' is not JSON serializable

- To serialize user-defined types we need to define encoding and decoding functions. This is shown in the following program where, we serialize **Complex** type.

```
import json

def encode_complex(x):
    if isinstance(x, Complex) :
        return(x.real, x.imag)
    else :
        raise TypeError('Complex object is not JSON serializable')

def decode_complex(dct):
    if '__Complex__' in dct :
        return Complex(dct['real'], dct['imag'])
    return dct

class Complex :
    def __init__(self, r = 0, i = 0) :
        self.real = r
        self.imag = i

    def print_data(self) :
        print(self.real, self.imag)

c = Complex(1.0, 2.0)
f = open('data', 'w+')
json.dump(c, f, default = encode_complex)
f.seek(0)
inc = json.load(f, object_hook = decode_complex)
print(inc)
```

- To translate a **Complex** object into JSON, we have defined an encoding function called **encode_complex( )**. We have provided this function to **dump( )** method's **default** parameter. **dump( )** method will use **encode_complex( )** function while serializing a **Complex** object.
- In **encode_complex( )** we have checked whether the object received is of the type **Complex**. If it is, then we return the **Complex** object data as a tuple. If not, we raise a **TypeError** exception.
- During deserialization when **load( )** method attempts to parse an object, instead of the default decoder we provide our decoder **decode_complex( )** through the **object_hook** parameter.

File and Directory Operations

- Python lets us interact with the underlying file system. This lets us perform many file and directory operations.
- File operations include creation, deletion, renaming, copying, checking if an entry is a file, obtaining statistics of a file, etc.
- Directory operations include creation, recursive creation, renaming, changing into, deleting, listing a directory, etc.
- Path operations include obtaining the absolute and relative path, splitting path elements, joining paths, etc.
- '.' represents current directory and '..' represents parent of current directory.
- Given below is a program that demonstrates some file, directory and path operations.

```
import os
import shutil

print(os.name)
print(os.getcwd( ))
print(os.listdir('.'))
print(os.listdir('..'))

if os.path.exists('mydir') :
    print('mydir already exists')
else :
```

```
    os.mkdir('mydir')

os.chdir('mydir')
os.makedirs('.\\dir1\\dir2\\dir3')
f = open('myfile', 'w')
f.write('Having one child makes you a parent...')
f.write('Having two you are a referee')
f.close( )
stats = os.stat('myfile')
print('Size = ', stats.st_size)

os.rename('myfile', 'yourfile')
shutil.copyfile('yourfile', 'ourfile')
os.remove('yourfile')

curpath = os.path.abspath('.')
os.path.join(curpath, 'yourfile')
if os.path.isfile(curpath) :
    print('yourfile file exists')
else :
    print('yourfile file doesn\'t exist')
```

P</> Programs

Problem 23.1

Write a program to read the contents of file 'messages' one character at a time. Print each character that is read.

Program

```
f = open('messages', 'r')
while True :
   data = f.read(1)
   if data == '' :
      break
   print(data, end = '')

f.close( )
```

Output

```
You may not be great when you start, but you need to start to be great.
Work hard until you don't need an introduction.
Work so hard that one day your signature becomes an autograph.
```

Tips

- **f.read(1)** reads 1 character from a file object **f**.
- **read( )** returns an empty string on reaching end of file.
- if **end = ''** is not used in the call to **print( )**, each character read will be printed in a new line.

Problem 23.2

Write a program that writes four integers to a file called 'numbers'. Go to following positions in the file and report these positions.

10 positions from beginning
2 positions to the right of current position
5 positions to the left of current position
10 positions to the left from end

Program

```
f = open('numbers', 'wb')
f.write(b'231')
f.write(b'431')
f.write(b'2632')
f.write(b'833')
f.close( )
f = open('numbers', 'rb')
f.seek(10, 0)
print(f.tell( ))
f.seek(2, 1)
print(f.tell( ))
f.seek(-5, 1)
print(f.tell( ))
f.seek(-10, 2)
print(f.tell( ))
```

```
f.close( )
```

Output

```
10
12
7
1
```

Problem 23.3

Write a Python program that searches for a file, obtains its size and reports the size in bytes/KB/MB/GB/TB as appropriate.

Program

```
import os

def convert(num) :
   for x in ['bytes', 'KB', 'MB', 'GB', 'TB'] :
      if num < 1024.0 :
         return "%3.1f %s" % (num, x)
      num /= 1024.0

def file_size(file_path) :
   if os.path.isfile(file_path) :
      file_info = os.stat(file_path)
      return convert(file_info.st_size)

file_path = r'C:\Windows\System32\mspaint.exe'
print(file_size(file_path))
```

Output

```
6.1 MB
```

Problem 23.4

Write a Python program that reports the time of creation, time of last access and time of last modification for a given file.

Program

```
import os, time

file = 'sampledata'
print(file)

created = os.path.getctime(file)
modified = os.path.getmtime(file)
accessed = os.path.getatime(file)

print('Date created: ' + time.ctime(created))
print('Date modified: ' + time.ctime(modified))
print('Date accessed: ' + time.ctime(accessed))
```

Output

```
sampledata
Date created: Tue May 14 08:51:52 2019
Date modified: Tue May 14 09:11:59 2019
Date accessed: Tue May 14 08:51:52 2019
```

Tips

- Functions **getctime( ), getmtime( )** and **getatime( )** return the creation, modification and access time for the given file. The times are returned as number of seconds since the epoch. Epoch is considered to be 1st Jan 1970, 00:00:00.
- **ctime( )** function of **time** module converts the time expressed in seconds since epoch into a string representing local time.

[A] State whether the following statements are True or False:

(a) If a file is opened for reading, it is necessary that the file must exist.

(b) If a file opened for writing already exists, its contents would be overwritten.

(c) For opening a file in append mode it is necessary that the file should exist.

[B] Answer the following questions:

(a) What sequence of activities take place on opening a file for reading in text mode?

(b) Is it necessary that a file created in text mode must always be opened in text mode for subsequent operations?

(c) While using the statement,

```
fp = open('myfile', 'r')
```

what happens if,

- 'myfile' does not exist on the disk
- 'myfile' exists on the disk

(d) While using the statement,

```
f = open('myfile', 'wb')
```

what happens if,

- 'myfile' does not exist on the disk
- 'myfile' exists on the disk

(e) A floating-point list contains percentage marks obtained by students in an examination. To store these marks in a file 'marks.dat', in which mode would you open the file and why?

[C] Attempt the following questions:

(a) Write a program to read a file and display its contents along with line numbers before each line.

(b) Write a program to append the contents of one file at the end of another.

(c) Suppose a file contains student's records with each record containing name and age of a student. Write a program to read these records and display them in sorted order by name.

(d) Write a program to copy contents of one file to another. While doing so replace all lowercase characters with their equivalent uppercase characters.

(e) Write a program that merges lines alternately from two files and writes the results to new file. If one file has less number of lines than the other, the remaining lines from the larger file should be simply copied into the target file.

(f) Suppose an **Employee** object contains following details:

employee code, employee name, date of joining, salary

Write a program to serialize and deserialize this data.

(g) A hospital keeps a file of blood donors in which each record has the format:

Name: 20 Columns
Address: 40 Columns
Age: 2 Columns
Blood Type: 1 Column (Type 1, 2, 3 or 4)

Write a program to read the file and print a list of all blood donors whose age is below 25 and whose blood type is 2.

(h) Given a list of names of students in a class, write a program to store the names in a file on disk. Make a provision to display the $\mathbf{n}^{th}$ name in the list, where **n** is read from the keyboard.

(i) Assume that a Master file contains two fields, roll number and name of the student. At the end of the year, a set of students join the class and another set leaves. A Transaction file contains the roll numbers and an appropriate code to add or delete a student.

Write a program to create another file that contains the updated list of names and roll numbers. Assume that the Master file and the Transaction file are arranged in ascending order by roll numbers. The updated file should also be in ascending order by roll numbers.

(j) Given a text file, write a program to create another text file deleting the words "a", "the", "an" and replacing each one of them with a blank space.