# Data I/O

Vivek K. S., Deepak G.

Information Systems Decision Sciences (ISDS)
MUMA College of Business
University of South Florida
Tampa, Florida

2017

# Memory Management

- There are two types of memory, the RAM memory and Disk memories.
- RAM is fast, expensive and volatile.
- Disk can hold more data, slower than RAM, cheap and non-volatile.
- Programming requires persistence, that is storing and retrieving of data using non-volatile disks such as disks.

# File I/O

- The simplest way to persist data is to use a flat file. This is just a sequence of bytes stored under a filename.
- Python's file operations are modeled on the familiar and popular Unix equivalents and it helps us to read from/write into a file in memory.
- The general syntax is fileobj = open(filename, mode)
- Here, mode is a string indicating the file's type and what you want to do with it.

# Mode

The first letter of mode indicates the operation

- r means read.
- w means write. If the file doesn't exist, its created. If the file does exist, its overwritten.
- x means write, but only if the file does not already exist.
- a means append (write after the end) if the file exists.

The second letter of mode is the file's type:

- t (or nothing) means text.
- b means binary.

# Writing a Flat file

```
review = "I have a few leave Bluetooth and \
I'm gonna say this really brings the BOOM.\
My music is bumping the colors are amazing \
and it also plays FM radio how amazing is that?!"

review_file = open('review','wt')
review_file.write(review)
review_file.close()
```

The write() function returns the number of bytes written. It does not add any spaces or newlines, as print() does.

# Writing Large files in Chunks

If we have a large source string, we can also write chunks until the source is done:

```
def write_in_chunks(text,chunk):
    fout = open('output_file', 'wt')
    size = len(text)
    offset = 0
    chunk = chunk
    while True:
        if offset > size:
            break
        fout.write(text[offset:offset+chunk])
        offset += chunk
    fout.close()
```

# Preventing Overwriting of Important Data

If we want to prevent overwriting of our data, we can prevent it with "xt". Let's implement that with a try..except block.

```
try:
        fout = open('output_file','xt')
        fout.write('Lets overwrite your data')
except FileExistsError:
        print('The file already exists!.
        That was a close one.')

'The file already exists!.
        That was a close one.'
```

# Read a file using Read() in Chunks

We can provide a maximum character count to limit how much read() returns at one time because read() method reads at one go and that could strain the memory.

```
review = ''
fin = open('output_file', 'rt')
chunk = 100
while True:
    fragment = fin.read(chunk)
    if not fragment:
        break
    review += fragment
fin.close()
len(review)
```

# Reading Using readline()

We can also read one line at a time using readline()

```
review = ''
fin = open('review', 'rt' )
while True:
    line = fin.readline()
    if not line:
        break
    review += line

fin.close()
```

# Reading Using Iterator

The easiest way is perhaps to use an iterator.

```
review = ''
fin = open('review', 'rt' )
for line in fin:
    print(line)
    review += line

fin.close()
```

# Closing Files with With

- In Python, if we forget to close a file that we have opened, it will be automatically closed if the file operation is happening as part of a function call.
- In that case, the file is closed when the function execution stops.
- If otherwise, the file has to be closed explicitly, and in this case we can use "with" to accomplish automatic closing of the file.
- Python has context managers to clean up things such as open files.

```
with open('review', 'wt') as fout:
    fout.write(poem)
```

# Seek and Tell

- Python keeps track of where we are whenever we read and write in file operations.
- The tell() returns the current offset from the beginning of the file in bytes.
- Seek() lets us jump to another byte offset in the file.

```
fin = open('review', 'rt')
fin.tell()
```

- Lets use seek to read some particular byte:

```
fin = open('review', 'rt')
fin.seek(50) # reading from the byte 50
```

# Complete Coding Exercise

Complete coding exercise is available at
`https://github.com/vivek14632/Python-Workshop/tree/master/Introducing%20Python/Chapter%206`

# Summary

- We learned how flat files can be used to create and store quick data.
- We learned how to perform different file operations such as read, write and use of iterators to read and write data.
- We learned the use of 'with' for automatically close files that may accidentally be left open.
- We learned how to prevent overwriting of critical data using exception mechanism.