Scripting & Computer Environments

Core Python: File Objects

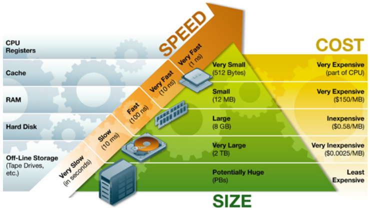
IIIT-H

A File

- A sequence of bytes stored on your computer or network.
- A named storage object managed by your OS.
- File objects are Python code's interface to external files on your system.
- Text files vs binary files

Memory Basics

- Memory Hierarchy

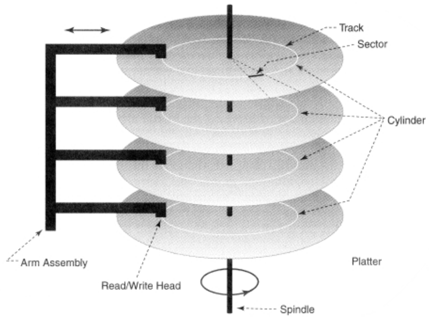


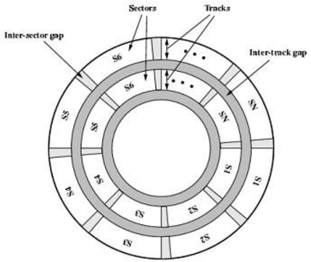
- Memory Access Modes
  - Linear
  - Random

Memory Basics: Disk I/O

- Operations (seek, read, write)

  e.g. Scenario of the booting process, opening file, saving file

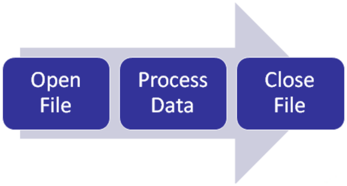




- Delays (seek time, rotational latency, data transfer)

File Operations (Python)

- Similar to C/C++, but easier in Python.
- The `file` object provides methods to manipulate files.
- Generic steps:



- Common Operations:
  1. Reading from a file
  2. Writing to a file

File Operations: 1. Opening

Syntax: `file_object = open(file_name, [access_mode])`

- `open()` returns an object of type *file* on a success, error otherwise.
- The returned `file_object` does not hold the file contents, rather a ‘window’ through which `file_name` can be viewed.

- Access modes:

Mode	**Operation**
r	open for read (default)
w	open for write
a	open for append
[rwa]+	open for read and write
[rwa]b	open for binary read, write & append respectively.
[rwa]b+	open for binary read and write
rU/U	open for read with universal Newline support.

File Operations: 2. Reading

- File be opened first for reading.
- Methods are accessed via a file object, say f.

Read Methods

```
f.read()                # read entire file & return as string
f.read(n)               # read n bytes
f.readline()            # read a line until '\n'
f.readlines()           # returns the file as a list
```

Example

```
>>>f=open('file1', 'r')          >>>f=open('file2')
>>>type(f)                       >>>line=f.readline()
>>>f.read()                      >>>print line
>>>f.close()                     >>>f.close()
```

File Operations: 2. Reading

- File be opened first for reading.
- Methods are accessed via a file object, say f.

Read Methods

```
f.read()           # read entire file & return as string
f.read(n)          # read n bytes
f.readline()       # read a line until ‘\n’
f.readlines()      # returns the file as a list
```

Example

```
>>>f=open(‘file1’, ‘r’)          >>>f=open(‘file2’)
>>>type(f)                       >>>line=f.readline()
>>>f.read()                      >>>print line
>>>f.close()                     >>>f.close()
```

File Operations: 3. Writing

- File be opened first for writing/appending.

Write Methods

```
f.write(str)                # write str unto the opened file
f.writelines(list)         # write strings in list as lines
```

Example

```
>>>f=open(‘somefile.txt’, ‘w’)
>>>f.write(‘CCC TGT GGA GCC ACA CCC TAG \n’)
>>>f.write(‘The curious CASE of BioInformatics \n’)
>>>f.close()

>>>open(‘somefile.txt’).read()                # reading in one go
>>>print open(‘somefile.txt’).read()          # Difference?
```

- Why close?
  - Open files consume resources
  - Shared access issues
- The Python garbage collector closes when reference count $= 0$.
- Good habit to close when done.
- The `close()` method frees the lock held by the file, if any.

  Syntax: `file_object.close()`
- Any operation on a closed file?

Other Useful Methods

- `f.seek(offset, [from])`
  - Move to `<offset>` bytes starting from position `<from>` within the file `f`.
  - `<from>` = 0 for beginning of file, 1 for current location, 2 for EOF.
- `f.tell()`
  - The current location in the open file `f`.

Example

```
>>>f=open('MyFile.txt', 'w+')
>>>f.write('Ancient of Days')
>>>f.tell()
>>>f.seek(5,0)
>>>f.seek(10,1)
>>>f.tell()
>>>f.readline()   # begins from?
>>>f.close()
```

File Attributes

- Hold auxiliary data related to the file object, f.
  - `f.name`
  - `f.mode`
  - `f.closed`

Example

```
>>>fo=open('MyFile.txt', 'r')
>>>print "Name: ", fo.name
>>>print "Mode: ", fo.mode
>>>print "File closed, right? ", fo.closed
```

Examples

1. Pickling/unpickling (serialization) of objects
2. Total number of lines or the first N lines of a file
3. Shortest/Longest line of a file
4. Copy file contents to another file.
5. Given a file `marks.txt` with the following data:

   ```
   Alice        30      20        30
   Bob          15      28        30
   Kumar        35      20.5      32
   ```

   Write a script that reads the marks from the file, computes the total mark for each person and writes to a file named `TotalMarks.txt`.

Working with System Files

- The sys module provides system-specific info related to your Python interpretter.

Example

```
dir(sys)

sys.platform                    # where is it installed?

sys.version                     # of Python interpretter

sys.prefix                      # the directory prefix

sys.argv                        # list of command-line args

sys.path                        # search path for modules
```

- The os module provides methods to use OS-dependent functionalities.

  1. Directory and File Manipulation

Example

```
# directory operations

x=os.getcwd()

os.listdir(x)

os.mkdir(‘somedir’)

os.rename(‘old’, ‘new’)

os.rmdir(‘somedir’)
```

```
# file operations

f=open(‘test.txt’)

f.close()

os.remove(‘test.txt’)

etc...
```

2. Executing System Commands (`os.system`)

Example

```
os.system('ls *')
os.system('cp source dest')                          etc...
```

3. Path Manipulation (`os.path`)

Example

```
p=os.path.abspath('test.txt')

os.path.split(p)

os.path.dirname(p)

os.path.basename(p)

os.path.join('path1', 'path2', 'path3')          etc...
```