Persistent Storage

ENGF0002 Design and Professional Skills

Mark Handley, University College London
based in part in slides by George Danezis

Term 1, 2019

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Persistent storage with files.

- **File systems** provide persistent storage, and expose it through a hierarchy of **directories** and **files**.
- A file is a **sequence of bytes** written to persistent memory (disk or other).
- Files support **read** and **write** operations on arbitrary positions of this sequence, as well as **updating** and **appending** data at the end of files.

Python supports handling files in `text mode` or `binary mode`. The first takes care of encoding of stings semi-automatically, while the second is more flexible but requires manual conversion of everything to bytes.

The open function.

The `open` function takes as parameters a file path and a mode, and returns a file object that may be used to `read` or `write`. All files should be **closed** after processing.

Modes include a combination of text ('t') or binary ('b'); read ('r'), write and truncate ('w') or append ('a'); and optionally update ('+').

The full documentation for `open`:
`https://docs.python.org/3/library/functions.html#open`

Reading all data from a text file.

```
>>> f = open(r"src/names.txt", "rt", encoding="utf8")
>>> data = f.read()  # Read full contents
>>> data.split()     # Split on new lines
['Ania', 'Jamie', 'Vasilis', 'Alberto', 'Mustafa', 'Marios', 'Rapha
>>> f.close()        # Close the file
```

Calling `read()` reads the full contents of the file into memory.
Remember to **close files** to free system resources.

Reading and writing text files.

```
11  def test_write():
12      f = open(r"src/store.txt", "wt")
13      f.write("Hello\n")
14      f.write("World\n")
15      f.close()
16
17      f2 = open(r"src/store.txt", "rt")
18      for line in f2:
19          print(line)
20      f2.close()
```

- Opening a file with ‘w’ **truncates** the file. (‘r+’ to update; ‘a’ to append.)
- A file is an **iterator**. The `for` loop reads each line in turn.
- Closing files can get tedious.

The ‘with’ control structure.

Acquiring and freeing resources is a very common pattern. The with control structure removes the need for manual closing of files, or freeing resources.

```
22  def test_write_with():
23      with open(r"src/store.txt", "wt") as f:
24          f.write("Hello\n")
25          f.write("World\n")
26
27      with open(r"src/store.txt", "rt") as f2:
28          for line in f2:
29              print(line)
```

Working with binary files.

You can use **seek** and **tell** to select a position in the file.

```
31  def test_binary():
32      with open("src/store.bin", "wb") as f:
33          f.write(b"\x00\x01\x02\x04")
34
35      with open("src/store.bin", "r+b") as f2:
36          f2.seek(2)              # goto byte 2.
37          b = f2.read(1)          # read 1 byte
38          assert b == b"\x02"
39          assert f2.tell() == 3 # new position
```

Standard input and output.

Programs have a standard input and standard output that act as files for reading and writing data.

```
43  def capitalize_input():
44      import sys
45      line = sys.stdin.readline()
46      cap_line = line.upper() + "\n"
47      sys.stdout.write(cap_line)
```

```
58  if __name__ == "__main__":
59      capitalize_input()
```

```
$ echo "Hello" | python src/open.py
HELLO
```

How to test programs that access the environment.

'**Monkeypatching**' is a test tactic in which you **substitute functions that access the environment** with known 'dummy' ones.

```
49  def test_stdio(monkeypatch):
50      # Monkeypatch the environment
51      monkeypatch.setattr(sys.stdin, 'readline', lambda: "A line")
52      def mock_write(text):
53          assert text == "A LINE\n"
54      monkeypatch.setattr(sys.stdout, 'write', mock_write)
55
56      capitalize_input()
```

You can monkey patch to **inject known data** pretending to be the environment. Or to **test interactions** with the environment. Tests should not access the environment.

Dealing with a lot of data.

Big data: Some processing tasks involve datasets that are **larger than the working memory** of a single computer.

- Reading the full file in memory is not possible.
- Instead **on-line algorithms** must be implemented to **work incrementally**.
- Read small chunks of data; process; write small chunks of output.

Sorting files on disk

Early computers had a tiny amount of working memory, and used tapes as persistent storage. Mergesort was especially important since it may be used to sort files in an incremental way with access to very little memory.

The social importance of sorting and indexing.

Modern search engines, need to find information within milliseconds!

Fast information retrieval is achieved through **indexing**:

- The file to be indexed is scanned linearly, and a list of tuples of (word, byte position) are extracted.
- The tuples are then sorted to produce an **inverted index**.
- The inverted index is stored to disk alongside the file.
- To find all instances of a term binary search is used on the inverted index to **retrieve the positions** in the file, if any, the term appears.

The second ingredient of search engines is **ranking**, that selects which files are most relevant for a search term.

Searching, Indexing, Files & Privacy.

Data have **meaning**, and can impact people's lives.
Protecting **people's rights**, including **privacy**, part of professional conduct.

UK Data Protection Act and the new General Data Protection Regulations:

- Defines 'personal information' as anything relating to a living individual.
- Lawful grounds (consent, contract, law) must exist to process.
- Can only store for defined purposes, time frames.
- Must be secured.
- Rights: access, remedy, to be forgotten.

`https://ico.org.uk/for-organisations/guide-to-data-protection/`

Technical implications of regulations.

- All personal information is mutable.
- Need to be able to find and remove it
  (Right to be Forgotten, Subject Access).
- That includes indexes and file backups.
- Notices and privacy statements need to be provided.
- No data processing should take place beyond what was notified.
- You need to use appropriate security controls.