More on String and File Processing

Marquette University



Problems with Line Endings

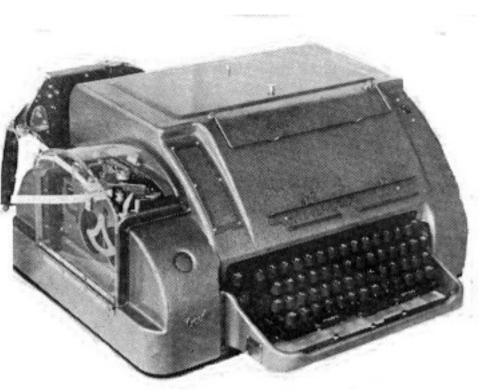
- ASCII code was developed when computers wrote to teleprinters.
 - A new line consisted of a carriage return followed or preceded by a line-feed.
- UNIX and windows choose to different encodings
 - Unix has just the newline character "\n"
 - Windows has the carriage return: "\r\n"
- By default, Python operates in "universal newline mode"
 - All common newline combinations are understood
 - Python writes new lines just with a "\n"
- You could disable this mechanism by opening a file with the universal newline mode disabled by saying:
 - open("filename.txt", newline='')

- Information technology has developed a large number of ways of storing particular data
 - Here is some background

			SPORD R.		
Case Data	Boot Sector NTF	54			
Pin Egi	00000000 00000000 00000000 00000000	0 1 2 3 4 5 6 7 4 65 6D 70 6C 61 74 65 65 63 74 6F 72 20 4E 54 2F 20 54 65 6D 70 6C 61	8 9 A 8 C D E F 20 22 42 6F 6F 74 20 53 46 53 22 00 0A 0D 0A 2F 74 65 20 62 79 20 53 74	Hesplate 'Boot S ector NTFS' / / Tesplate by St	Juregidee Boot Sector NTFS (a) D'Downloads'wintex
	00000030 00000040 00000050	65 66 61 68 20 46 6C 65 0D 04 2F 2F 20 50 20 57 77 61 72 65 20 54 65 63	69 73 63 68 6D 61 6E 6E 61 79 73 20 53 6F 66 74		File size: 1,61 1,614 byt
	00000060	41 47 0D 0A 0D 0A 2F 2F 70 70 6C 69 65 64 20 74	20 54 6F 20 62 65 20 61 6F 20 73 65 63 74 6F 72	AG // To be a pplied to sector	DOSname: BOOTSE~3.1
	00000000	20 30 20 6F 66 20 61 6E 72 6D 61 74 74 65 64 0D	20 4E 54 46 53 2D 66 6F 04 2F 2F 20 6C 6F 67 69	0 of an NTFS-fo reatted // logi	Default Edit Mode State: orig
	000000000	63 61 6C 20 64 72 69 76 74 60 65 20 6D 69 72 72 6F 66 20 74 60 65 20 62	65 20 6F 72 20 74 6F 20 6F 72 20 63 6F 70 79 20 6F 6F 74 00 04 2F 2F 20	cal drive or to the airror copy of the boot //	Unde-level Unde-revenses
	00000000	73 65 63 74 6F 72 2C 20 6C 6C 20 62 65 20 6C 6F	77 68 69 63 68 20 77 69 63 61 74 65 64 20 6E 65	mector, which wi 11 be located ne	Oreation time: 27-03-2 10.35
	000000F0 00000100 00000110	61 72 20 74 68 65 20 65 2F 20 74 68 65 20 70 61 0D 0A 0D 0A 64 65 73 63	6E 64 20 6F 66 0D 0A 2F 72 74 69 74 69 6F 6E 2E 72 69 70 74 69 6F 6E 20	ar the end of / / the partition description	Last write time: 21-03-2 16:04
	00000120 00000130 00000140	22 42 6E 6E 74 20 73 65 61 6E 20 4E 54 46 53 20 6E 22 0D 0A 61 70 70 6C	63 74 6F 72 20 6F 66 20 70 61 72 74 69 74 69 6F 69 65 73 5F 74 6F 20 64	"Boot sector of an NTF5 partitio	Abributes kone
	03000150 03000160 03000170 03000180	69 73 68 0D 0A 73 65 63 6E 65 64 0D 0A 0D 0A 72 30 78 30 38 09 22 45 42 2F 20 62 79 74 65 73 20	74 6F 72 2D 61 6C 69 67 65 71 75 69 72 65 73 20 22 20 09 09 09 09 09 2F 30 20 74 6F 20 32 20 61	ned requires 0x00 "EB" / / bytes 0 to 2 a	Note hexadeo Drancfer art: CP 1. Offsatz hexadeo Bytes per page 32x16-
	03030140 03030140 03030180	72 65 0D 0A 72 65 71 75 32 09 22 39 30 22 09 09 50 20 69 68 73 74 72 75		re requires 0x0 2 '90' // JX P instruction r	Window #. No. of sendows:
	00000100 00000100 000001E0 000001F0	65 71 75 69 72 65 73 20 20 35 34 20 34 36 20 35 20 49 44 20 6D 75 73 74 53 22 0D 04 72 65 71 75	30 70 30 33 09 22 34 45 33 20 32 30 22 09 2F 2F 20 62 65 20 22 4E 54 46 69 72 65 73 20 30 78 31	equires 0x03 '4E 54 46 53 20' // ID must be 'MTF 5' requires 0x1	Opticent evaluation TEMP folder: 21.0.081 C1/Deen/ Bert AppCate/Local/Te
	Page 1 of 4	Offset	0. +116 1		n/a Sar

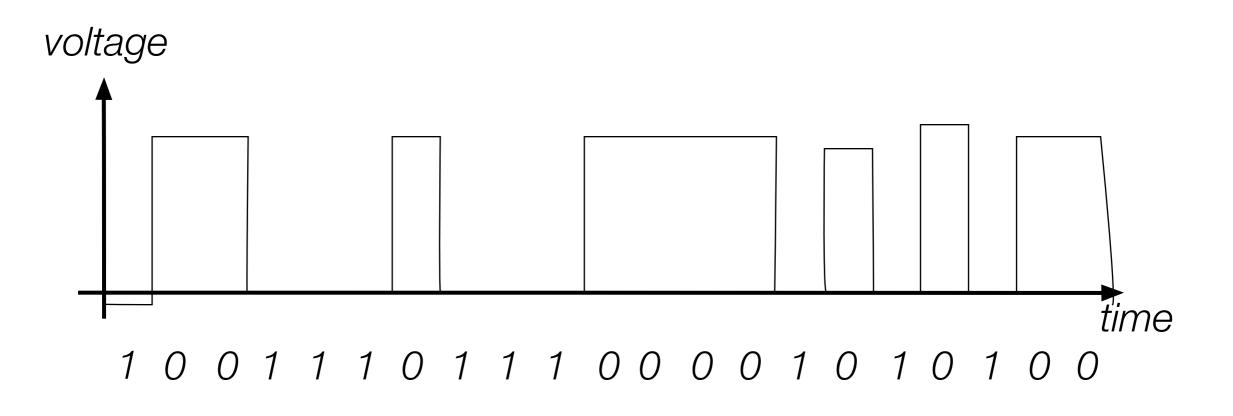
Using a forensics tool (Winhex) in order to reveal the bytes actually stored

• Teleprinters



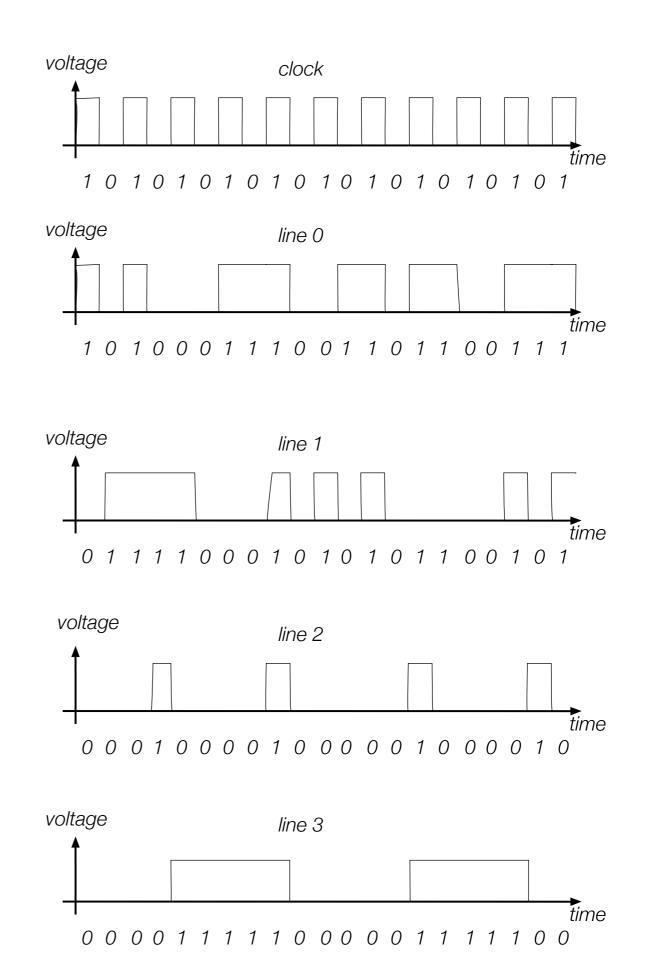
- Used to send printed messages
 - Can be done through a single line
 - Use timing to synchronize up and down values

- Serial connection:
 - Voltage level during an interval indicates a bit
 - Digital means that changes in voltage level can be tolerated without information loss



- Parallel Connection
 - Can send more than one bit at a time
 - Sometimes, one line sends a timing signal

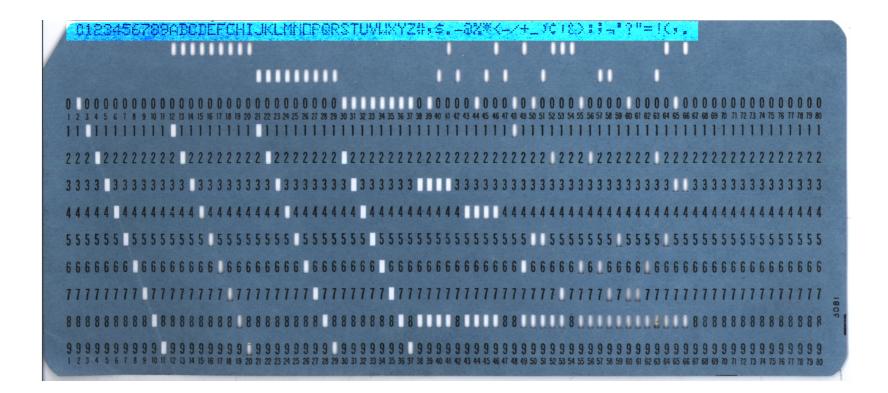
- Sending
 - 1000
 - 0100
 - 1100
 - 0100
 - ...
- Small errors in timing and voltage are repaired automatically



- Need a code to transmit letters and control signals
- Émile Baudot's code 1870
 - 5 bit code
 - Machine had 5 keys, two for the left and three for the right hand
 - Encodes capital letters plus NULL and DEL
 - Operators had to keep a rhythm to be understood on the other hand

- Many successors to Baudot's code
 - Murray's code (1901) for keyboard
 - Introduced control characters such as Carriage Return (CR) and Line Feed (LF)
 - Used by Western Union until 1950

- Computers and punch cards
 - Needed an encoding for strings
 - EBCDIC 1963 for punch cards by IBM
 - 8b code



- ASCII American Standard Code for Information Interchange 1963
 - 8b code
 - Developed by American Standard Association, which became American National Standards Institute (ANSI)
 - 32 control characters
 - 91 alphanumerical and symbol characters
 - Used only 7b to encode them to allow local variants
 - Extended ASCII
 - Uses full 8b
 - Chooses letters for Western languages

- Unicode 1991
 - "Universal code" capable of implementing text in all relevant languages
 - 32b-code
 - For compression, uses "language planes"

- UTF-7 1998
 - 7b-code
 - Invented to send email more efficiently
 - Compatible with basic ASCII
 - Not used because of awkwardness in translating 7b pieces in 8b computer architecture

- UTF-8 Unicode
 - Code that uses
 - 8b for the first 128 characters (basically ASCII)
 - 16b for the next 1920 characters
 - Latin alphabets, Cyrillic, Coptic, Armenian, Hebrew, Arabic, Syriac, Thaana, N'Ko
 - 24b for
 - Chinese, Japanese, Koreans
 - 32b for
 - Everything else

	S sign	E: Biased Exponent 8 bits	Z Significand 23 bits		
Bit index • Doubl		31 30 2322 e precision format			
	S	E: Biased Exponent 11 bits	Z Significand 20 bits		

- Numbers
 - There is a variety of ways of storing numbers (integers)
 - All based on the binary format
 - For floating point numbers, the exact format has a large influence on the accuracy of calculations
 - All computers use the IEEE standard

Python and Encodings

- Python "understands" several hundred encodings
 - Most important
 - ascii (corresponds to the 7-bit ASCII standard)
 - **utf-8** (usually your best bet for data from the Web)
 - latin-1
 - straight-forward interpretation of the 8-bit extended ASCII
 - never throws a "cannot decode" error
 - no guarantee that it read things the right way

Python and Encodings

- If Python tries to read a file and cannot decode, it throws a decoding exception and terminates execution
- We will learn about exceptions and how to handle them soon.
- For the time being: Write code that tells you where the problem is (e.g. by using line-numbers) and then fix the input.
- Usually, the presence of decoding errors means that you read the file in the wrong encoding

Using the os-module

- With the os-module, you can obtain greater access to the file system
 - Here is code to get the files in a directory

```
import os
def list_files(dir_name):
    files = os.listdir(dir_name)
    for my_file in files:
        print(my_file, os.path.getsize(dir_name+"/"+my_file))
```

list_files("Example")

Using the os-module

import os

Get a list of file names in the directory

def list_files(dir_name):
 files = os.listdir(dir_name)
 for my_file in files:
 print(my_file, os.path.getsize(dir_name+"/"+my_file))

list_files("Example")

import os

```
def list_files(dir_name):
    files = os.listdir(dir_name)
    for my_file in files:
        print(my_file, os.path.getsize(dir_name+"/"+my_file))
```

list files("Example")

Creating the path name to the file

import os

```
def list_files(dir_name):
    files = os.listdir(dir_name)
    for my_file in files:
        print(my_file, os.path.getsize(dir_name+"/"+my_file))
```

list files("Example")

Gives the size of the file in bytes

import os

```
def list_files(dir_name):
    files = os.listdir(dir_name)
    for my_file in files:
        print(my file, os.path.getsize(dir_name+"/"+my_file))
```

list files("Example")

List and

- Output:
 - Note the Mac-trash file

```
RESTART: /Users/thomasschwa
le14/generator.py
.DS_Store 6148
results1.csv 384
results0.csv 528
results2.csv 432
results3.csv 368
results4.csv 464
```

- Using the listing capability of the os-module, we can process all files in a directory
 - To avoid surprises, we best check the extension
 - Assume a function process_a_file
 - Our function opens a comma-separated (.csv) file
 - Calculates the average of the ratios of the second over the first entries

1.290, 12.495 2.295, 11.706 3.063, 9.083

8.833

9.733

10.032

33.335

47.130

93.389

623

797

094 306 047

502 782

953 131 514

827 687

452 849 109

786

103.726

7, 50.559

27.295 3, 62.268

41

20

73

39

78

17

22

55

4.058,

9.373 5, 1.858, 14.439 5, 15.820 21.861 19.097

10.838 8,

0.280

37.029

37.459

34.994

37.458

66.393

62.255

0.929,

1.147, 1.093

1.997, 8.833

2.781, 10.032

4.225, 9.733

5.455, 15.820

6.151, 20.939

6.573, 26.547

8.058, 33.335

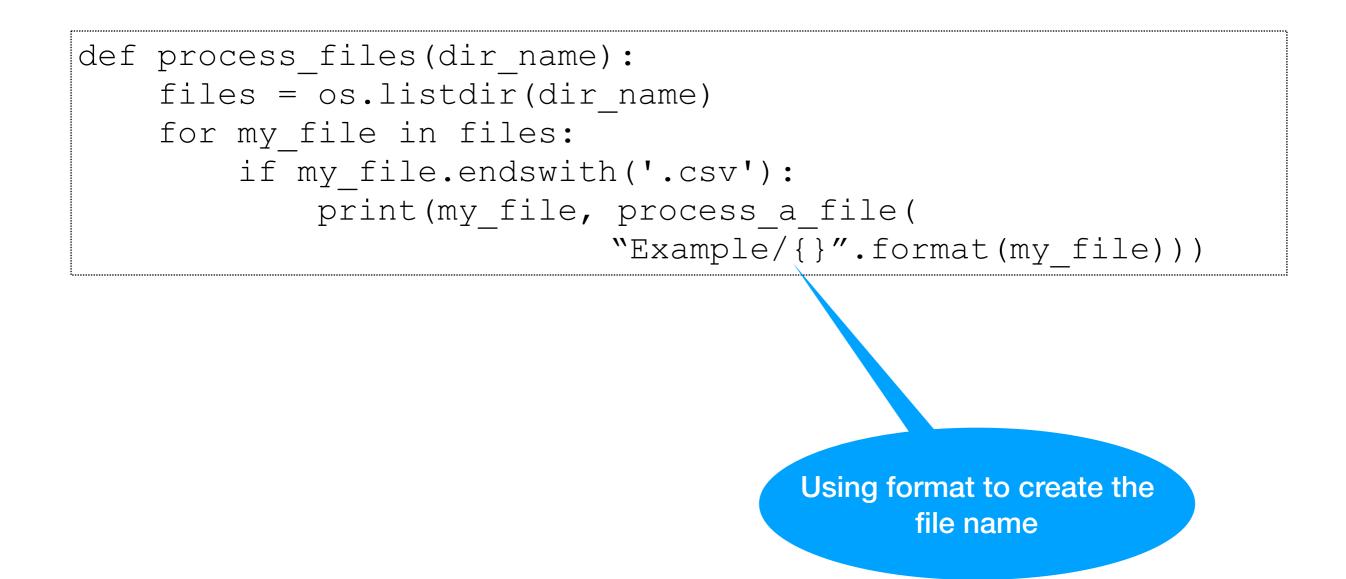
9.132, 37.546

10.474, 47.130 84.116 6

- The process_a_file takes the file-name
 - Calculates the average ratio

```
def process a file(file name):
    with open(file name, "r") as infile:
        suma = 0
        nr lines = 0
        for line in infile:
            nr lines+=1
            array = line.split(',')
            suma+= float(array[1])/float(array[0])
    return suma/nr lines
```

- To process the directory
 - Get the file names using os
 - For each file name:
 - Check whether the file name ends with .csv
 - Call the process_a_file function
 - Print out the result



RESTART: /Users/thomasschwarz/Docu le14/generator.py >>> process_files('Example') results1.csv 5.2819632072675295 results0.csv 5.920382285263983 results2.csv 5.7506863373894666 results3.csv 4.801235259621119 results4.csv 6.409464135625922