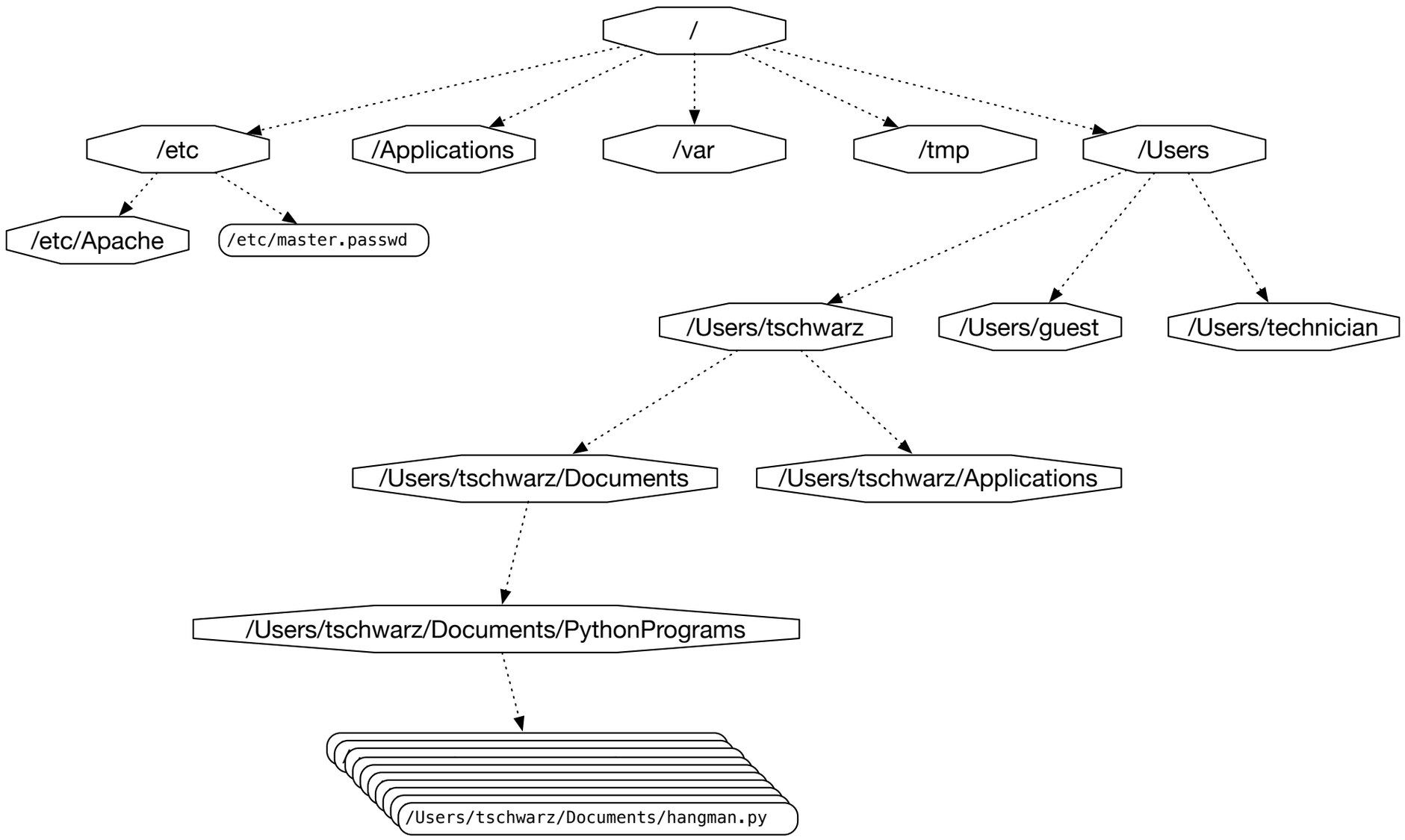
Dealing with Files

Thomas Schwarz, SJ

Files

- Files
  - Basic container of data in modern computing system
  - Organized into a hierarchy of directories

Files



A small subset of directories and files on a system

Files in Python

- Access to file system through os module
  - Discussed later in course
- Files accessed in
  - text mode
    - Contents interpreted according to encoding
  - binary mode
    - Contents not interpreted

Files in Python

- Python interacts by files through
  - reading
  - writing / appending
  - both

Files in Python

- Files need to be opened
  - File given by name
    - Relative path: Navigation from directory of the file
    - Absolute path: Navigation from the root of the file system

Files in Python

- File Name Examples:
  - Absolute path on a Mac / Unix

```
/Users/tjschwarzsj/Google Drive/AATeaching/Python/Programs/pr.py
```

  - Relate path on a Mac / Unix
    - “../“ means move up on directory

```
pr.py

../Slides/week7.key
```

Files in Python

- Windows uses backward slashes to separate directories in a file name
  - Sometimes need to be escaped: \\
  - Absolute paths need to include drive name:
    - c:\\users\\tschwarz\\My Documents\\Teaching\\temp.py
- *We will typically read and create files in the same directory as the python program is located*

Files in Python

- Before files are used, program needs to open them
- After they are being used, program should close them
  - Will automatically closed when program terminates
  - Long-running programs could hog resources

Opening Files in Python

- File objects have normal variable names

```
inFile = open(“data.txt”,”w”)
```

- opens a file “data.txt” in write mode

- open takes :
  - file name — absolute / relative path
  - mode — r (read), w (write), a (appending)
  - mode — b (binary), “” (not binary)

Closing Files in Python

- We close file by invoking close
  - `inFile.close()`

Why we need to close files

- Files are automatically closed when the program terminates
- When one application has opened a file for writing it acquires a write lock on the file and no other application can access the file.
- When one application has opened a file for reading, it acquires a read lock on the file and no other application can write to it.
- If you write programs that last more than a few seconds, you do not want to hog files when you do not need them.

With-clauses

- Python 3 allows us to open and close files in a single block (context)

```
with open("twoft8.11.txt") as inFile, open("twoftres8.11.txt",
"w") as outFile:
    #Here you work with the file
```

Processing Files in Python

- We write strings to the file

```
with open('somefile.txt','wt') as f:
 f.write(str(500)+"\n")
```

- Redirect print

```
with open('somefile.txt','wt') as f:
 print(500, file = f)
```

Processing Files in Python

- Reading files
  - The read-instruction

```
string = inFile.read(10)
```

reads ten bytes of the file

- Read the entire file

```
with open('somefile.txt', 'rt' as f:
    data = f.read()
```

Processing Files in Python

- Reading files
  - Read line by line

```
with open('somefile.txt', 'rt') as f:
    for line in f:
        #process line
```

More String Processing

- To process read lines:
  - `strip()` and its variants `lstrip(), rstrip()`
    - Remove white spaces (default) or list of characters from the beginning & end of the string
  - `split()` creates a list of words separated by white space (default)

```
"This is a sentence with many words in it.”.split()
```

```
['This', 'is', 'a', 'sentence', 'with', 'many', 'words', 'in', 'it.']
```

Examples

- Finding all words over 13 letters long in “Alice in Wonderland”
  - Download from Project Gutenberg

```
import string

with open("alice.txt", "rt", encoding = "utf-8") as f:
    for line in f:
        for word in line.split():
            if len(word) > 13:
                print(word)
```

Examples

- Count the number of words and of lines in “Alice in Wonderland”
  - Read the file line by line
  - The number of words in a line is the length of line.split.

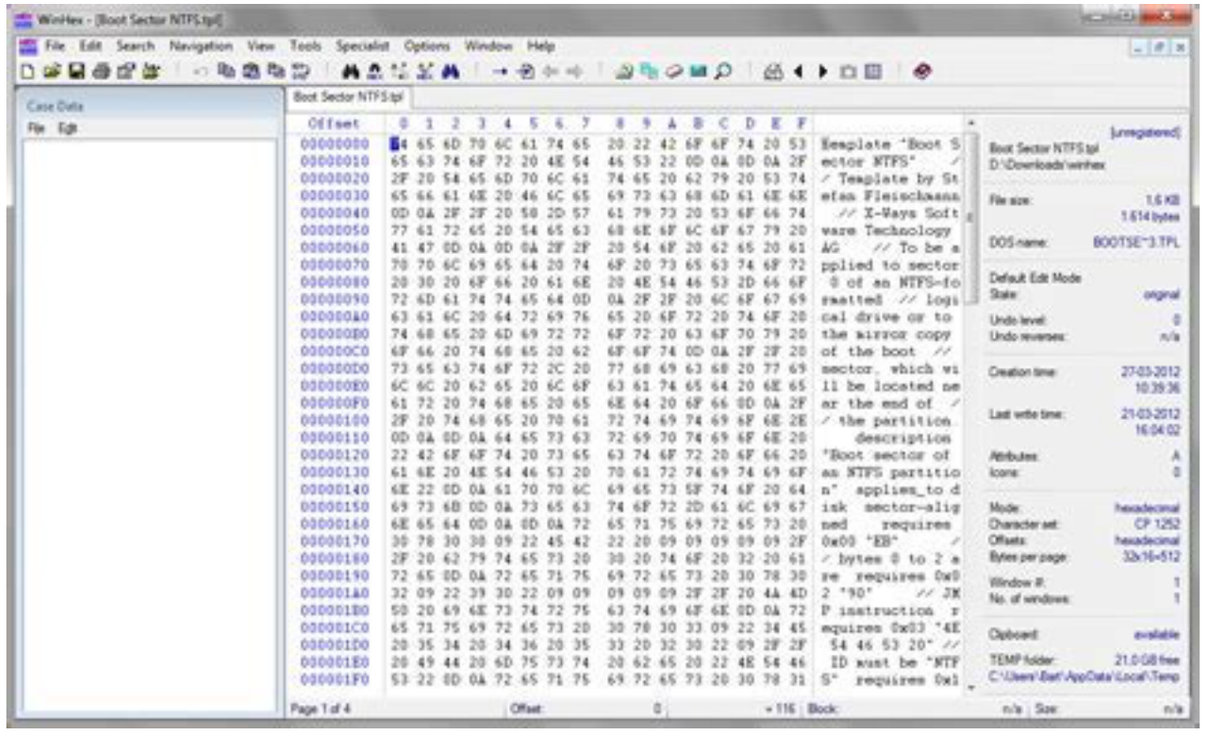
```
import string

line_counter = 0
word_counter = 0
with open("alice.txt", "rt", encoding = "utf-8") as f:
    for line in f:
        line_counter += 1
        word_counter += len(line.split())
print(line_counter, word_counter)
```

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=‘’)`

Encodings

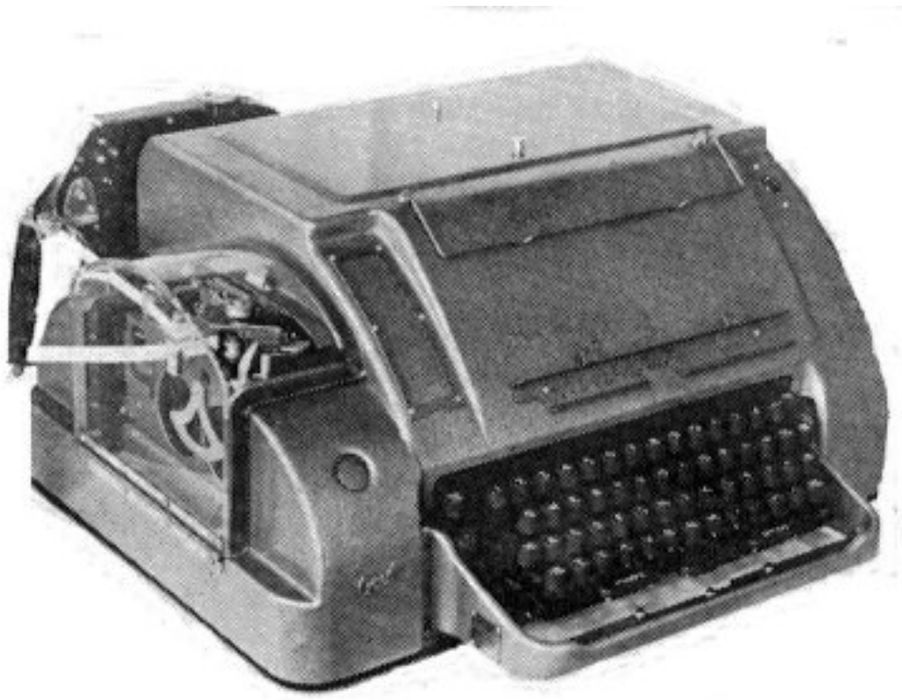
- Information technology has developed a large number of ways of storing particular data
  - Here is some background



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

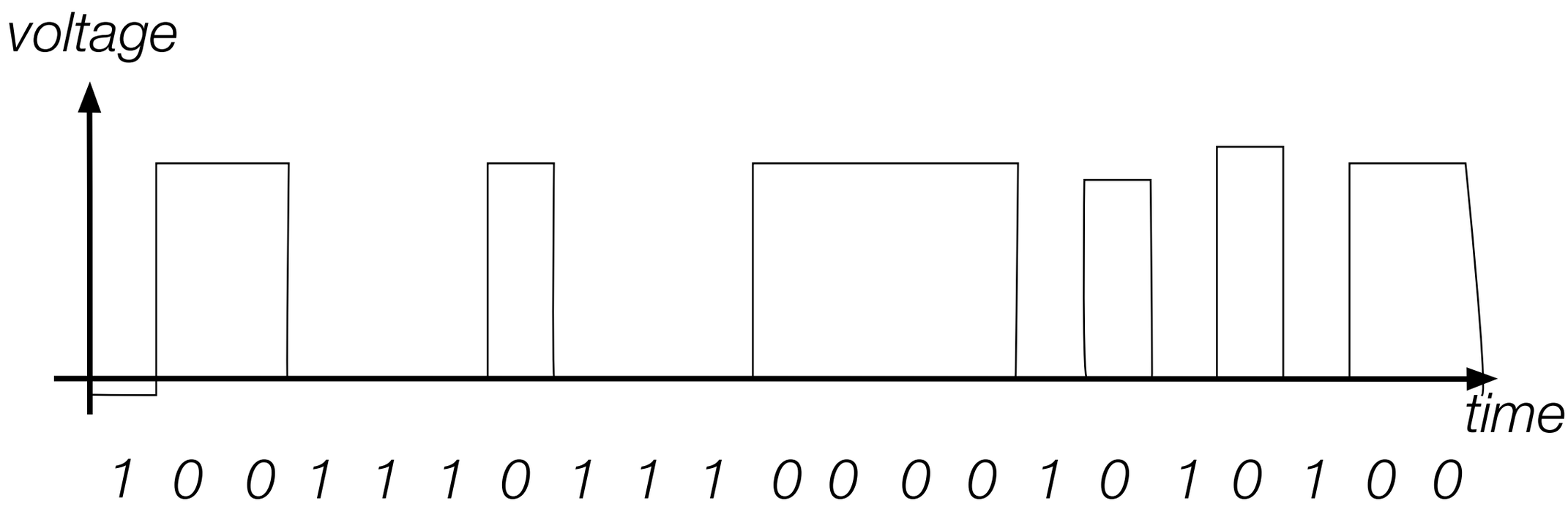
Encodings

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



Encodings

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

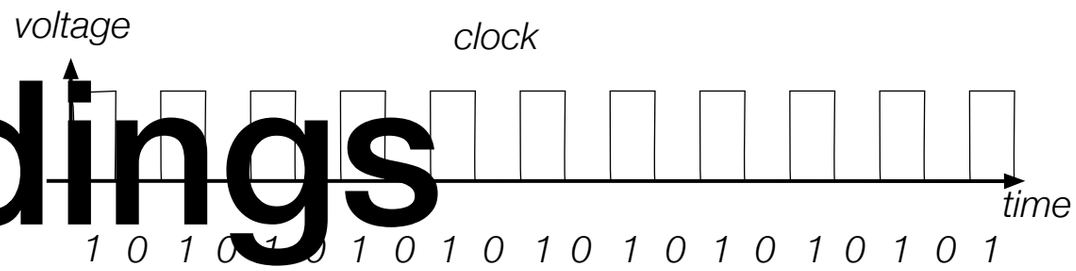


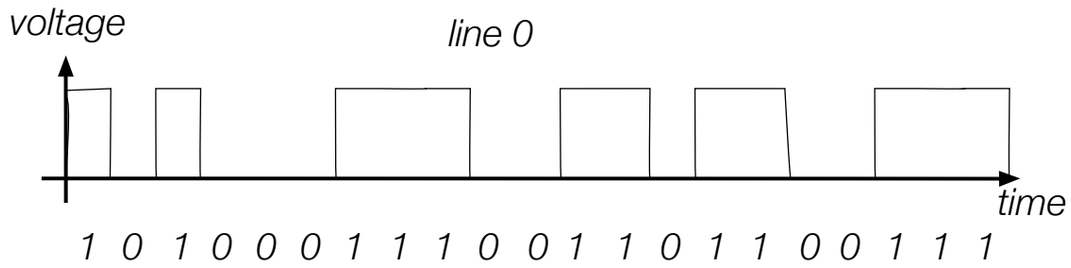
Encodings

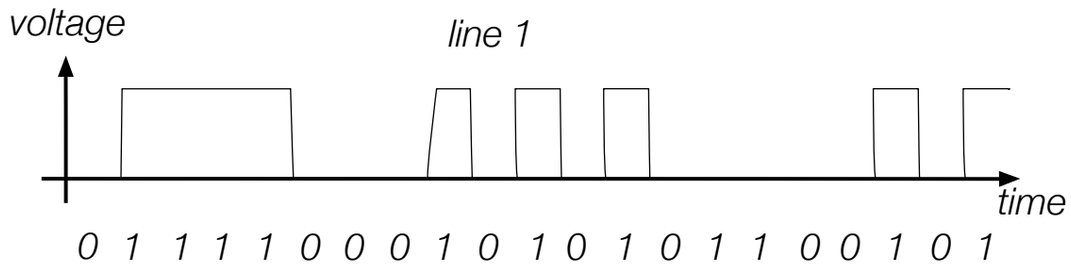
- Parallel Connection
  - Can send more than one bit at a time
  - Sometimes, one line sends a timing signal

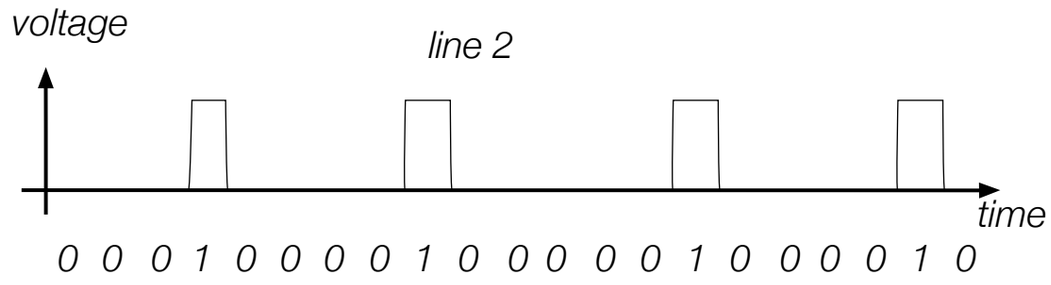
Encodings

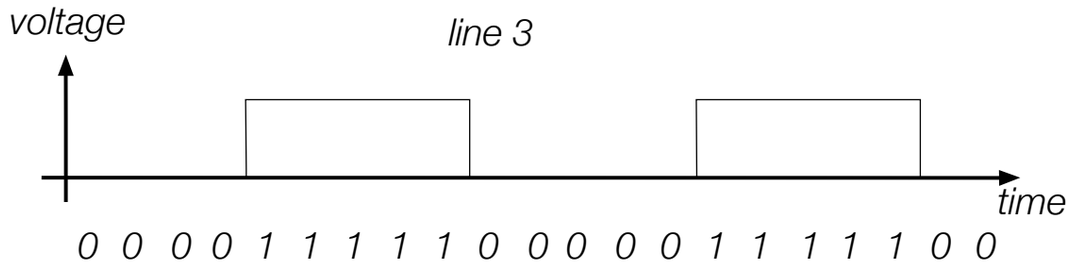
- Sending
  - 1000
  - 0100
  - 1100
  - 0100
  - ...
- Small errors in timing and voltage are repaired automatically











Encodings

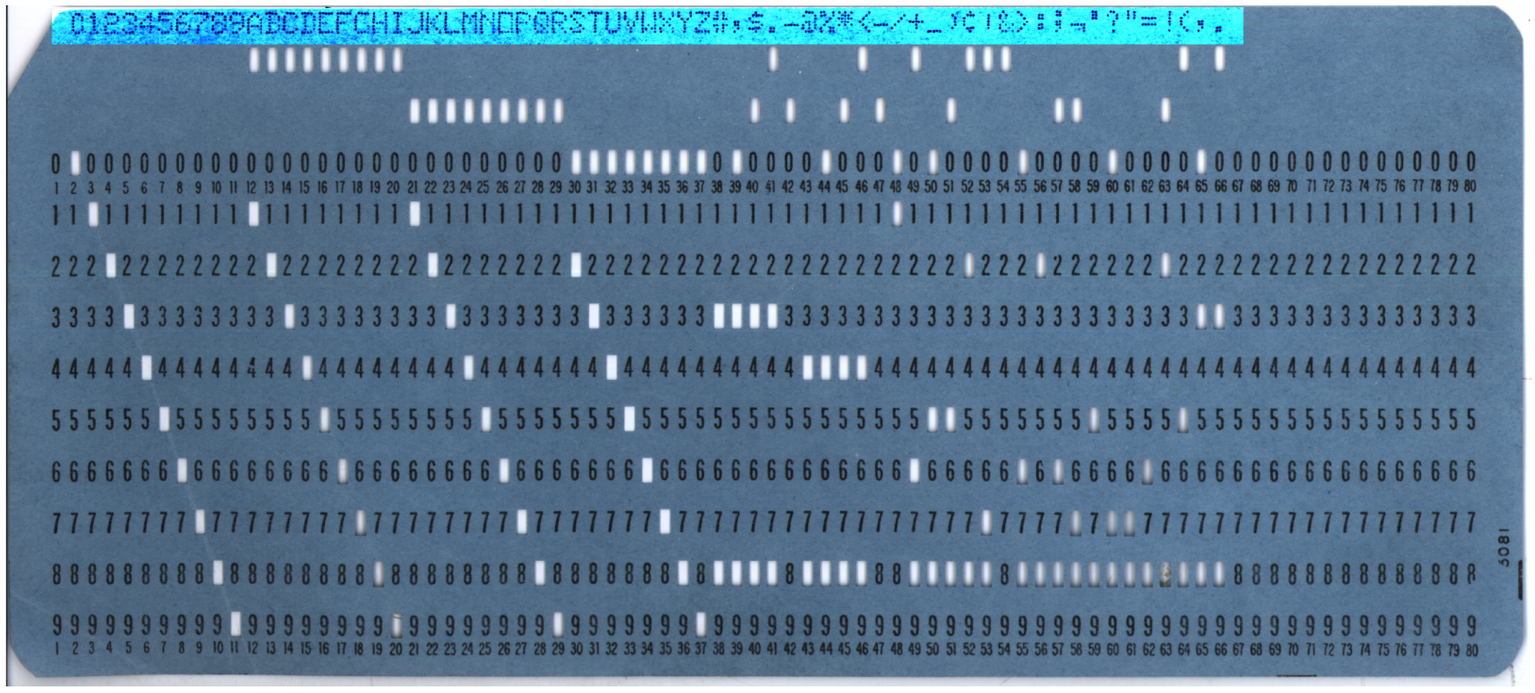
- Need a code to transmit letters and control signals
- Émile Baudot's code 1870
  - 5 bit code
  - Machine had 5 keys, two for the left and three for the right hand
  - Encodes capital letters plus NULL and DEL
  - Operators had to keep a rhythm to be understood on the other side

Encodings

- 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

Encodings

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



Encodings

- 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

Encodings

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

Encodings

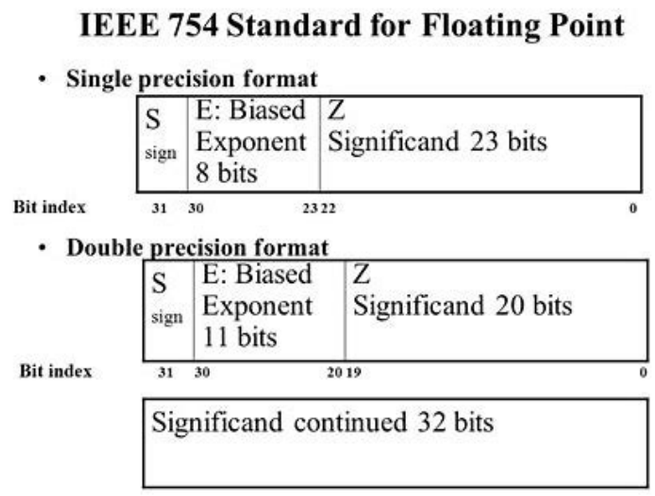
- UTF-7 — 1998
  - 7b-code
    - Invented to send email more efficiently
    - Compatible with basic ASCII
    - Not used because of awkwardness in translating 7b pieces in 8b computer architecture

Encodings

- 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

Encodings

- 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

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))
```

Get a list of file names in the directory

```
list_files(“Example")
```

Use the os-module

```
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

Use the os-module

```
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

Use the os-module

```
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

Use the os-module

- 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
```

Use the os-module

- 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

Use the os-module

- 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
```

Use the os-module

- 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

Use of the os-module

```
def process_files(dir_name):
    files = os.listdir(dir_name)
    for my_file in files:
        if my_file.endswith('.csv'):
            print(my_file, process_a_file(
                "Example/{}".format(my_file)))
```

Using format to create the file name

Use of the os-module

```
 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
```

Encodings

- Whenever you see strings:
  - Think about encoding and decoding
    - Example: the ë
      - `'ë'.encode('utf-8').decode('latin-1')`
    - gives
      - `'Ã«'`
- Mixing encodings often creates chaos

Encodings

- Python is very good at guessing encodings
  - Do not guess encodings
    - E.g.: Processing html: read the http header:
      - `Content-Type: text/html; charset=utf-8`
  - If you need to guess, there is a module for it:
    - `chardet.detect(some_bytes)`

Encodings

- Thinking about encoding and decoding string allows easy internationalization

Bytearrays

- On (rare) occasions, you might want to work with bytes directly
  - Read the file in binary mode
  - Bytearray allows you to manipulate directly binary data
    - bytes have range 0-255
- `content = bytearray(infile.read())`