W4 D1

Fonts

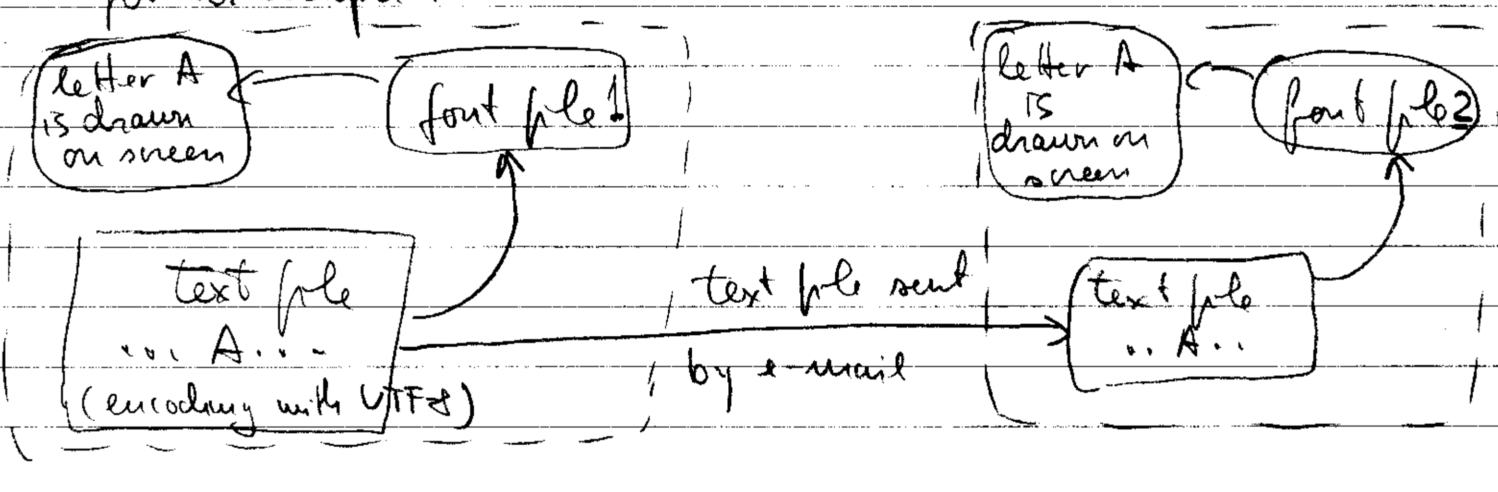
- Text is encoded with binary symbols using standards such as ASCII or Unicode (UTF-8)
  How is text displayed on screen?

- Font file = contains instructions to draw letters on screen

Alphabet letter encoding (ex UTF-8)	Drawing instructions
101... (letter A)	(How to draw A)
⋮	

Font file = a table that matches binary encodings of letters with a "drawing"

- Font files can be very large; they are a resource that belongs to a computer (more precisely, to the operating system) and are not transmitted with the text when you are sending e-mail for example.



- Font files have names
  eg: Times New Roman
  Arial
  Courier
  ⋮
  → names identify the general look of the letters (a font family)

  → Font family contains several standard looks of the font → Roman / Italic / Bold face

  $\underbrace{\text{Roman, Italic, Bold face}}$ all have some common characteristics, that's why they are one family.

- Different types of font families → serif fonts (eg Times new roman)
  → sans serif (eg Arial)
  → monospace (eg Courier)

Experiments

- type text in Word Document. Change the font to "Webdings" for ex. What happened?

- Web-pages: google for "Unicode" / main page. select "What is unicode" on left.
  → you can look @ pages written in many languages using different scripts.
  - issues: font availability / text encodings

Representing numbers

- representing # is less problematic

There is a natural, mathematical way of translating numbers to a binary encoding without the need of complicated standards & tables

→ base 2 numbering system.

Usual numbers (base 10)

$$756 = 7 \cdot 100 + 5 \cdot 10 + 6 = (700 + 50 + 6) =$$
$$= 7 \cdot 10^2 + 5 \cdot 10^1 + 6 \cdot 10^0.$$

↓ nice representation for humans with 10 fingers.

1011110100 is 756 written in base 2
(a sequence of 2 symbols that can be represented / stored in a computer).

$$1011110100 = 1 \cdot 2^9 + 0 \cdot 2^8 + 1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 +$$
$$+ 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0$$

Simple examples

Base 2	Base 10
1	1
10	2
11	3
100	4
1000000000	512 (2^9)

"important (or easy)" numbers

Base 2	Base 10	
0	0	
1	1	
10	2	(2^1)
100	4	(2^2)
1000	8	(2^3)
10000	16	(2^4)
100000	32	(2^5)

$5 = ?$

$5 = 4 + 1 = 1 \cdot 2^2 + 1 \cdot 2^0 = (101)_{\text{base } 2}$ you do not add 2

$7 = \underbrace{4 + 2 + 1}_{} = (111)_{\text{base } 2}$

try to express it as a sum of ~~powers~~ distinct powers of two

Why this method and not ASCII or Unicode?
Arithmetic operations can be performed directly on the binary encoded sequences

$$\begin{array}{r} 111 + \\ 101 \\ \hline 1100 \end{array}$$

Rules for addition are the same!

$$\begin{array}{r} 7 + \\ 5 \\ \hline 12 \end{array}$$

base 10

Simple rules $1 + 0 = 0 + 1 = 1$

$1 + 1 = 10$

$1 + 1 + 1 = 11$

(Recall how to add 2 numbers from primary school)

Other useful ways for representing numbers (useful for humans when they need to deal with binary sequences)

Base 16 numbers (hex numbers)

(ex) $27 = 16 + 11$ (represent a # using powers of 16!)
$= (1B)_{\text{base } 16}$

$33 = 32 + 1 =$
$= 2 \cdot 16 + 1 = (21)_{\text{base } 16}.$

Hex	Base 10
1	1
10	16
100	256
1000	4096

A → 10 B → 11 C → 12
D → 13 E → 14 F → 15

Benefit is not coming from converting base 10 numbers to base 16, but converting base 2 (long sequences of bits) to something easier to handle for humans.

Simple trick $10(11\,11)(0100)$ → group every 4 bits from right

$(2)_{10}$ $(15)_{10}$ (4) base 10
$\shortparallel$ $\shortparallel$ $\shortparallel$
$(2)_{16}$ $(F)_{16}$ (4) base 16

Converted hex representation of this long bit sequence is $(2F4)_{16}$.

each symbol is a group of 4 bits.

Other numbers

- negative integers (−34, etc)
  - chosen in such a way that subtractions are nothing but normal additions with a negative #.

- fractional numbers 2.74 are represented by 2 integers — 274 (mantissa) M
  — −2 (exponent) E

By convention, fractional numbers are given by $M \cdot 10^E$ ($274 \cdot 10^{-2} = 2.74$)
This is called a "floating point" representation of 2.74.

$\boxed{\text{OBS}}$ (take home message from this discussion):
Computers cannot represent all numbers, just (integers not too large or too small, some rational #). There is no way to represent irrational #s.
=> mathematical calculations of computers are imprecise.