How does the computer store numbers and other information?

Bits and Characters

- ASCII – American Standard Code for Information Interchange
- Character encoding scheme – each upper and lower case letter in the English alphabet and other characters such as # and $ represented as a sequence of 7 0s and 1s
- First introduced in the 1960s
- Today Universal Character Set (aka Unicode) is more common UTF-8, UTF-16 and UTF-32

Glyph	ASCII	Unicode
#	0010 0011	0000 0000 0010 0011
$	0010 0100	0000 0000 0010 0100
A	0100 0001	0000 0000 0100 0001
a	0110 0001	0000 0000 0100 0001
©		0000 0000 1010 1001
æ		0000 0000 1110 0110
Δ		0000 0011 1001 0100
α		0000 0011 1011 0001

ASCII and Unicode mappings are compatible for the 2^7 = 128 ASCII characters. The bottom 4 characters do not have encodings in ASCII

Bits and Bytes

- A **bit** is a single binary digit – 0 or 1
- Short for **b**inary dig**it**
- Tukey – Legendary statistician and the father of "Exploratory Data Analysis" coined the term
- A byte is a collection of 8 bits – 0001 0011

Usually written with a space between the two sets of 4 digits for readability

Representing Numbers

Recall that when we write a 3-digit number, e.g.,

105

We are using the decimal system and we mean:
1 hundred, **0** tens, **5** ones,

That is: $(1 * 10^2) + (0 * 10^1) + (5 * 10^0)$
where the digits range from 0, 1, 2, …, 9

Representing Numbers in Binary

- We can do the same to represent numbers in binary
- The binary number:

1101001

- Now we have powers of 2 and digits 0 and 1:

$(1 * 2^6) + (1 * 2^5) + (0 * 2^4) + (1 * 2^3)$
$+ (0 * 2^2) + (0 * 2^1) + (1 * 2^0)$

- In decimal this is 64+32+8+1 = 105!

What is the decimal value of the following 8-digit binary number?

00110001

Value	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
Position	7	6	5	4	3	2	1	0	
Base 2	0	0	1	1	0	0	0	1	00110001
Decimal	0	0	32	16	0	0	0	1	32+16+1 = 49

Different Types of Numbers

- Integer types are stored in the computer as described
- But what about numeric types, e.g.

0.25? Or -3.14? Or 1/3?

- Notice that the computer cannot store 1/3 because it only has so many digits to use
- The computer uses the notion of scientific notation to store numbers

Scientific Notation

- General form:

$$a * 10^b$$

a: mantissa   b: exponent

10: base   And sign +/-

0.023 -> 2.3 * 10^{-2}

-2100 -> -2.1 * 10^3

Double-Precision Floating Point

- 8 bytes (64 bits)
- Sign bit: 1 bit
- Exponent: 11 bits
- Mantissa/significand: 53 bits (stored as 52)

$$(-1)^{\text{sign}}(1.b_{51}b_{50}...b_0)_2 \times 2^{e-1023}$$

or

$$(-1)^{\text{sign}}\left(1+\sum_{i=1}^{52} b_{52-i}2^{-i}\right) \times 2^{e-1023}$$

How does this impact our work?

- There is a limit to how precisely we can represent numbers.
- Need to be aware of this when doing calculations.
- For example, in many cases it is better to do calculations on the log scale.
- Example: instead of multiplying two numbers, take sum of logs, then exponentiate back only when strictly necessary.

EXAMPLES IN R

Instead of 1/x where x=1.6* 10^308

Represent it as log(x) and –log(x)