

# TK1100

Tegnsett  
og  
(litt) om andre media

---

Det finnes 10 typer mennesker i verden,  
de som kan binært og de som ikke kan det.

# SIST

---




- Posisjontallsystemer
  - Basis, potens av basis
  - Konvertering
- (Primitive) **datatyper**
  - Heltall, (flyttall)
  - Binærtall og toerkomplement
  - Addisjon
    - Det CPUen bruker
  - Hexadesimale tall
    - Notasjon
  - Flyttall
    - Kodesystem, ikke «tallsystem»

# I DAG

---

- Koding av bokstaver (**alfanumeriske** tegn)
- Kode-tabeller
  - **ASCII** (7 bit)
  - Mac Roman, **ISO 8859-1**, Windows 1252
  - **UNICODE**
    - UTF-16
    - UTF-8
- Andre media (litt)
  - Filformater
  - Bilde og lyd
- Komprimering

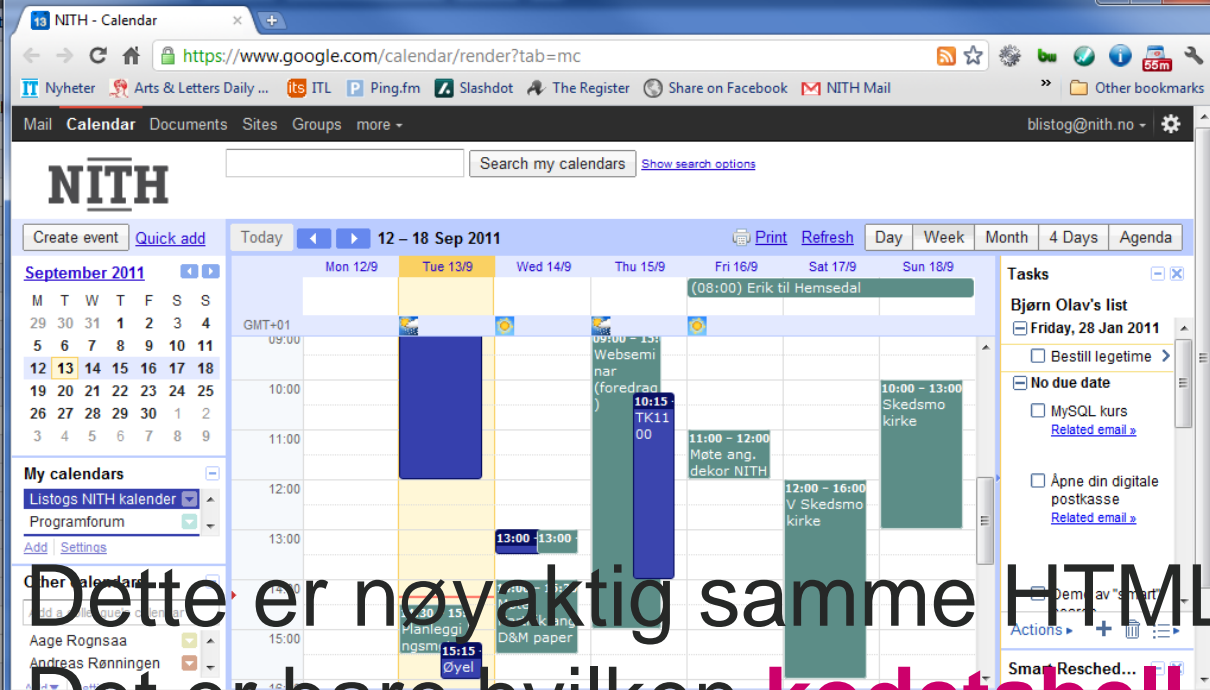
# Tolking av binære koder (noen få)

	0011 1110	0010 0000	0111 0010	0011 0111
(Hexadesimalt)	0x3E	0x20	0x72	0x37
32-bit heltall	1 042 313 783			
16-bit heltall	15 904		29 239	
32 bit flyttall	0.156686			
BCD	Umulig!	20	72	37
IPv4-adresse	62.32.114.55			
ASCII	>	mellomrom	r	7
Scankode(USB)			F23	
UTF-16	兆		爷	
JVM bytekode	istore_3	Istore_2	frem	Istore
X86 opkode	DS:	AND	JNO	AAA

- Hvordan vet man hvilken tolkning som er riktig?
- Det bestemmer (bruken i) programmet!

---

# **HVORFOR SKAL VI BRY OSS OM TEGNSETT?**



Encoding:  
UTF-8 -> UTF-16LE

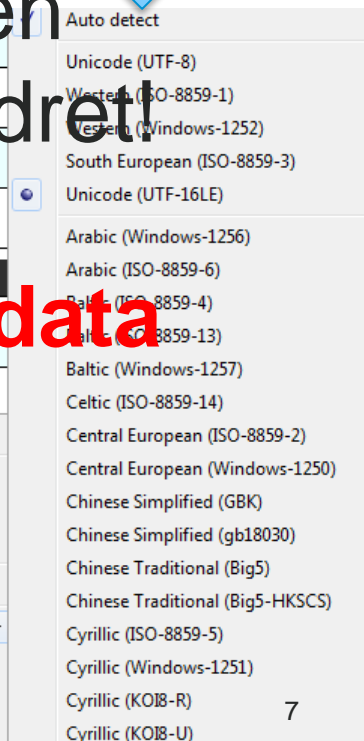
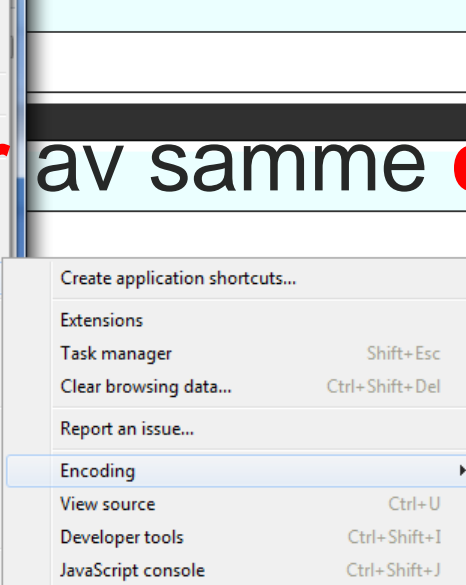
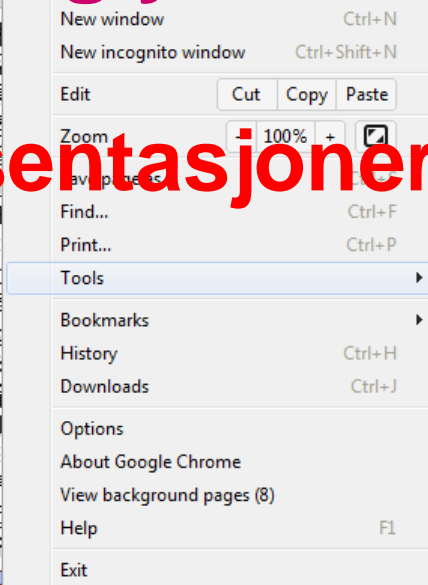


Dette er nøyaktig samme HTML/CSS-kode!

Det er bare hvilken **kodetabell** browseren

braker for å velge **glyfer** som som er endret

med. To representasjoner av samme data



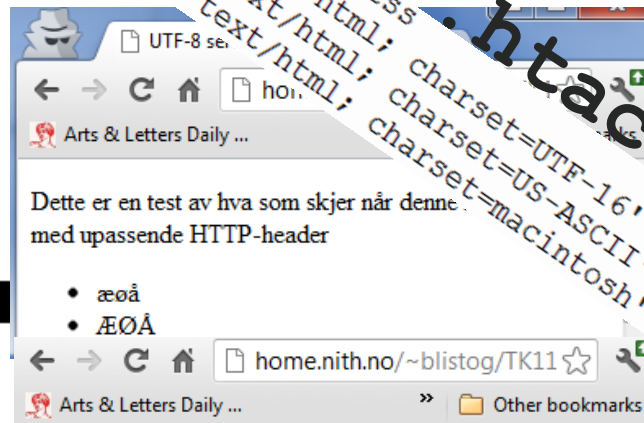
# Web og HTML

- Selve **HTML-koden** er en **ren text-fil**
  - Det kan gå (veldig) galt
  - **Browseren** bryr seg mest om hva **serveren** sier

GNU nano 2.2.4

File: test.html

```
<!DOCTYPE html>
<head>
<title>UTF-8 servert som UTF-8</title>
<meta charset="UTF-8">
</head>
<body>
<p>Dette er en test av hva som skjer når denne
filen leveres med upassende HTTP-header</p>
<ul>
<li>æøå</li>
<li>ÆØÅ</li>
</ul>
</body>
```



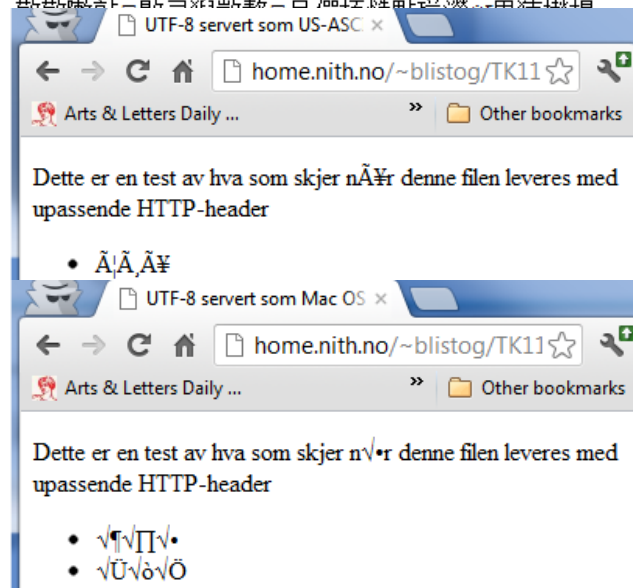
UTF-8 servert som US-ASC

Arts & Letters Daily ...

Other bookmarks

Dette er en test av hva som skjer nÅr denne filen leveres med upassende HTTP-header

- Å, Å, Å



ÅRSÅK: .htaccess

0v02->cat .htaccess

AddType 'text/html; charset=UTF-16' html1

AddType 'text/html; charset=UTF-ASCII' html2

AddType 'text/html; charset=macintosh' html3



# Moral (så langt)

---

- Text brukes til mye mer enn å bare lage/lagre bokstaver og utforme lesbare setninger
- Vi bruker det også til å kode andre former for adferd(program), data, struktur, utseende, m.m.
- Dersom man ikke passer på går det lett galt
  - Ulike systemer har ulike karakter-tabeller som default (standard)

# ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EH	SUB	ESC	FS	GS	RS	US
2		!	"	#	\$	%	&	'	(	)	+	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	-	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

- 7-bit
- 32 kontroll-tegn
- 95/96 alfa-nummeriske tegn
- Inntil 2007 (?) mest brukt på web!

# ASCII

# Binær Kode -> Kodetabell -> Glyf

## Glyf

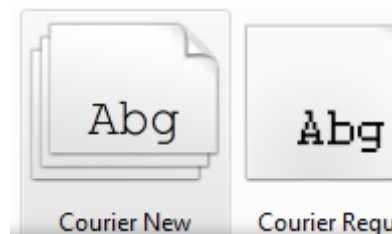
- Det du ser på skjermen er ikke en ASCII-character
- Det er et skjerm**bilde** som bestemmes av kodetabellen
- Betegnelse på et slikt skjerm**bilde** er **glyf**
- Hvordan den *ser ut* bestemmes av **font-tabellen**

0100 0001



	0	1	2	3
0	NUL	SOH	STX	ETX
1	DLE	DC1	DC2	DC3
2		!	"	#
3	0	1	2	3
4	@	A	B	C

ASCII-  
tabell



Font-  
Definisjonene  
(Operativsystemet)



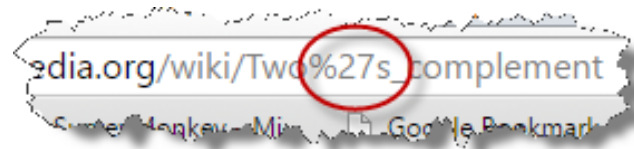
A A-glyfen

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	<b>NUL</b> (null)	32	20	040	&#32;	<b>Space</b>	64	40	100	&#64;	<b>@</b>	96	60	140	&#96;	<b>`</b>
1	1	001	<b>SOH</b> (start of heading)	33	21	041	&#33;	<b>!</b>	65	41	101	&#65;	<b>A</b>	97	61	141	&#97;	<b>a</b>
2	2	002	<b>STX</b> (start of text)	34	22	042	&#34;	<b>"</b>	66	42	102	&#66;	<b>B</b>	98	62	142	&#98;	<b>b</b>
3	3	003	<b>ETX</b> (end of text)	35	23	043	&#35;	<b>#</b>	67	43	103	&#67;	<b>C</b>	99	63	143	&#99;	<b>c</b>
4	4	004	<b>EOT</b> (end of transmission)	36	24	044	&#36;	<b>\$</b>	68	44	104	&#68;	<b>D</b>	100	64	144	&#100;	<b>d</b>
5	5	005	<b>ENQ</b> (enquiry)	37	25	045	&#37;	<b>%</b>	69	45	105	&#69;	<b>E</b>	101	65	145	&#101;	<b>e</b>
6	6	006	<b>ACK</b> (acknowledge)	38	26	046	&#38;	<b>&amp;</b>	70	46	106	&#70;	<b>F</b>	102	66	146	&#102;	<b>f</b>
7	7	007	<b>BEL</b> (bell)	39	27	047	&#39;	<b>'</b>	71	47	107	&#71;	<b>G</b>	103	67	147	&#103;	<b>g</b>
8	8	010	<b>BS</b> (backspace)	40	28	050	&#40;	<b>(</b>	72	48	110	&#72;	<b>H</b>	104	68	150	&#104;	<b>h</b>
9	9	011	<b>TAB</b> (horizontal tab)	41	29	051	&#41;	<b>)</b>	73	49	111	&#73;	<b>I</b>	105	69	151	&#105;	<b>i</b>
10	A	012	<b>LF</b> (NL line feed, new line)	42	2A	052	&#42;	<b>*</b>	74	4A	112	&#74;	<b>J</b>	106	6A	152	&#106;	<b>j</b>
11	B	013	<b>VT</b> (vertical tab)	43	2B	053	&#43;	<b>+</b>	75	4B	113	&#75;	<b>K</b>	107	6B	153	&#107;	<b>k</b>
12	C	014	<b>FF</b> (NP form feed, new page)	44	2C	054	&#44;	<b>,</b>	76	4C	114	&#76;	<b>L</b>	108	6C	154	&#108;	<b>l</b>
13	D	015	<b>CR</b> (carriage return)	45	2D	055	&#45;	<b>-</b>	77	4D	115	&#77;	<b>M</b>	109	6D	155	&#109;	<b>m</b>
14	E	016	<b>SO</b> (shift out)	46	2E	056	&#46;	<b>.</b>	78	4E	116	&#78;	<b>N</b>	110	6E	156	&#110;	<b>n</b>
15	F	017	<b>SI</b> (shift in)	47	2F	057	&#47;	<b>/</b>	79	4F	117	&#79;	<b>O</b>	111	6F	157	&#111;	<b>o</b>
16	10	020	<b>DLE</b> (data link escape)	48	30	060	&#48;	<b>0</b>	80	50	120	&#80;	<b>P</b>	112	70	160	&#112;	<b>p</b>
17	11	021	<b>DC1</b> (device control 1)	49	31	061	&#49;	<b>1</b>	81	51	121	&#81;	<b>Q</b>	113	71	161	&#113;	<b>q</b>
18	12	022	<b>DC2</b> (device control 2)	50	32	062	&#50;	<b>2</b>	82	52	122	&#82;	<b>R</b>	114	72	162	&#114;	<b>r</b>
19	13	023	<b>DC3</b> (device control 3)	51	33	063	&#51;	<b>3</b>	83	53	123	&#83;	<b>S</b>	115	73	163	&#115;	<b>s</b>
20	14	024	<b>DC4</b> (device control 4)	52	34	064	&#52;	<b>4</b>	84	54	124	&#84;	<b>T</b>	116	74	164	&#116;	<b>t</b>
21	15	025	<b>NAK</b> (negative acknowledge)	53	35	065	&#53;	<b>5</b>	85	55	125	&#85;	<b>U</b>	117	75	165	&#117;	<b>u</b>
22	16	026	<b>SYN</b> (synchronous idle)	54	36	066	&#54;	<b>6</b>	86	56	126	&#86;	<b>V</b>	118	76	166	&#118;	<b>v</b>
23	17	027	<b>ETB</b> (end of trans. block)	55	37	067	&#55;	<b>7</b>	87	57	127	&#87;	<b>W</b>	119	77	167	&#119;	<b>w</b>
24	18	030	<b>CAN</b> (cancel)	56	38	070	&#56;	<b>8</b>	88	58	130	&#88;	<b>X</b>	120	78	170	&#120;	<b>x</b>
25	19	031	<b>EM</b> (end of medium)	57	39	071	&#57;	<b>9</b>	89	59	131	&#89;	<b>Y</b>	121	79	171	&#121;	<b>y</b>
26	1A	032	<b>SUB</b> (substitute)	58	3A	072	&#58;	<b>:</b>	90	5A	132	&#90;	<b>Z</b>	122	7A	172	&#122;	<b>z</b>
27	1B	033	<b>ESC</b> (escape)	59	3B	073	&#59;	<b>:</b>	91	5B	133	&#91;	<b>[</b>	123	7B	173	&#123;	<b>{</b>
28	1C	034	<b>FS</b> (file separator)	60	3C	074	&#60;	<b>&lt;</b>	92	5C	134	&#92;	<b>\</b>	124	7C	174	&#124;	<b> </b>
29	1D	035	<b>GS</b> (group separator)	61	3D	075	&#61;	<b>=</b>	93	5D	135	&#93;	<b>]</b>	125	7D	175	&#125;	<b>}</b>
30	1E	036	<b>RS</b> (record separator)	62	3E	076	&#62;	<b>&gt;</b>	94	5E	136	&#94;	<b>^</b>	126	7E	176	&#126;	<b>~</b>
31	1F	037	<b>US</b> (unit separator)	63	3F	077	&#63;	<b>?</b>	95	5F	137	&#95;	<b>_</b>	127	7F	177	&#127;	<b>DEL</b>

# ASCII – struktur

PUGG!!

- **0x00-0x1F**: De 32 første kodene er **kontrolltegn**.
  - 0x07 er f.eks. beep i høyttaler
  - Linjeskift er
    - i **Windows** CarriageReturn (0x0D) **OG** LineFeed 0x0A),
    - i **Unix/Linux** og **OSX** bare LineFeed (0x0A)
  - Mellomrom-tegnet er **0x20**
    - Noensinne sett %20 e.l. i en URL?
- Tallene **kodes** med: **0x30-0x39**
  - Binærtallet **0000 0000** har ASCII-kode **0011 0000**
- "A" er **0x41** -> 01**00** 0001  
"a" er **0x61** -> 01**10** 0001
  - En enkelt bit forskjell mellom store og små bokstaver
  - Numerisk sorteres dermed alle de store foran de små bokstavene



# ANSI ESC-sekvenser og konsoll

- Windows konsollet (CMD.EXE) er det eneste som ikke støtter ANSI-escape sekvenser for å posisjonere og bruke farger.

```
public class Farger
{
    public static void main(String[] args)
    {
        //Sekvensene starter med ESC - som har ASCII-kode 27, som er 033
        //oktalt og 0x1B i hex; i Unix er oktalt vanlig brukt.
        //\033[2J Nuller skjermen
        System.out.print("\033[2J");
        //\033[13;40H plasser cursor ca midt i skjermbildet
        System.out.print("\033[13;40H");
        //\033[3x;4x sørger for text- og bakgrunnfarge
        //x-verdier 0-sort,1-rød,2-grønn,3-gul,4-blå,5-Magenta,6-Cyan,7-Hvit|
        //25;H er koden for å flytte cursor til bønn av 25x80 konsollet
        System.out.println("\033[37;44m Hello World\033[25;H");
        //Fargekoden må så nullstilles igjen
        System.out.print("\033[0;0m");
    }
}
```

Ov02->javac Farger.java

Ov02->java Farger



Hello World!

- I Linux og OSX er de derimot fremdeles støttet og brukt.

# ASCII: Styrker og svakheter

- Hvert tegn er alltid 1 byte
- Aldri noe problem med Big/Little Endian
- Enkelt å flytte på = interoperabelt
- Støtter bare «amerikansk» alfabet!
- Alternative nasjonsspesifikke kodetabeller (og fonter..) ble dermed nødvendige.
- I HTML løste man dette med egne koder for «særnorske bokstaver»
  - &AElig; er Æ, mens &aelig; er æ
  - &Oslash; er Ø, mens &oslash; er ø
  - &Aring; er Å, mens &aring; er å.

INTEROPERABELT

# WINDOWS 1252

## ISO 8859-1

- «Europeiske tegn» manglet i ASCII
- Beholdt ASCII og utvidet til 8 bit
- Mange forskjellige løsninger på 1980-tallet (Code tables)
- Standarden for Vest europeiske skulle være ISO 8859-1 (Latin-1)



# ISO 8859-1 (Latin-1) kodetabell

	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
0	NUL	DLE	space	0	@	P	`	p			no break space	°	À	Đ	à	ð
1	SOH	DC1	!	1	A	Q	a	q			ı	±	Á	Ñ	á	ñ
2	STX	DC2	”	2	B	R	b	r			¢	²	Â	Ò	â	ò
3	ETX	DC3	#	3	C	S	c	s			£	³	Ã	Ó	ã	ó
4	EOT	DC4	\$	4	D	T	d	t			¤	´	Ä	Ô	ä	ô
5	ENQ	NAK	%	5	E	U	e	u			¥	µ	Å	Õ	å	õ
6	ACK	SYN	&	6	F	V	f	v	un- defined			¶	Æ	Ö	æ	ö
7	BEL	ETB	'	7	G	W	g	w			§	·	Ç	×	ç	÷
8	BS	CAN	(	8	H	X	h	x			¨	¸	È	Ø	è	ø
9	HT	EM	)	9	I	Y	i	y			©	¹	É	Ù	é	ù
A	LF	SUB	*	:	J	Z	j	z			ª	º	Ê	Ú	ê	ú
B	VT	ESC	+	;	K	[	k	{			«	»	Ë	Û	ë	û
C	FF	FS	,	<	L	\	l				¬	¼	Ì	Ü	ì	ü
D	CR	GS	-	=	M	]	m	}			-	½	Í	Ý	í	ý
E	SO	RS	.	>	N	^	n	~			®	¾	Î	Þ	î	þ
F	SI	US	/	?	O	_	o	DEL			¯	¿	Ï	ß	ï	ÿ

# Windows 1252 kodetabell

Bygger på ISO 8859-1

	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
0	NUL	DLE	space	0	@	P	`	p	€	undef	no break space	°	À	Đ	à	ð
1	SOH	DC1	!	1	A	Q	a	q	undef	‘	¡	±	Á	Ñ	á	ñ
2	STX	DC2	”	2	B	R	b	r	,	’	¢	²	Â	Ò	â	ò
3	ETX	DC3	#	3	C	S	c	s	f	“	£	³	Ã	Ó	ã	ó
4	EOT	DC4	\$	4	D	T	d	t	„	”	¤	´	Ä	Ô	ä	ô
5	ENQ	NAK	%	5	E	U	e	u	...	•	¥	µ	Å	Õ	å	õ
6	ACK	SYN	&	6	F	V	f	v	†	–	¦	¶	Æ	Ö	æ	ö
7	BEL	ETB	'	7	G	W	g	w	‡	—	§	·	Ç	×	ç	÷
8	BS	CAN	(	8	H	X	h	x	^	~	¨	¸	È	Ø	è	ø
9	HT	EM	)	9	I	Y	i	y	‰	™	©	¹	É	Ù	é	ù
A	LF	SUB	*	:	J	Z	j	z	Š	š	ª	º	Ê	Ú	ê	ú
B	VT	ESC	+	;	K	[	k	{	‹	›	«	»	Ë	Û	ë	û
C	FF	FS	,	<	L	\	l		Œ	œ	¬	¼	Ì	Ü	ì	ü
D	CR	GS	-	=	M	]	m	}	undef	undef	-	½	Í	Ý	í	ý
E	SO	RS	.	>	N	^	n	~	Ž	ž	®	¾	Î	Þ	î	þ
F	SI	US	/	?	O	_	o	DEL	undef	ÿ	¯	¸	Ï	ß	ï	ÿ

# UNICODE

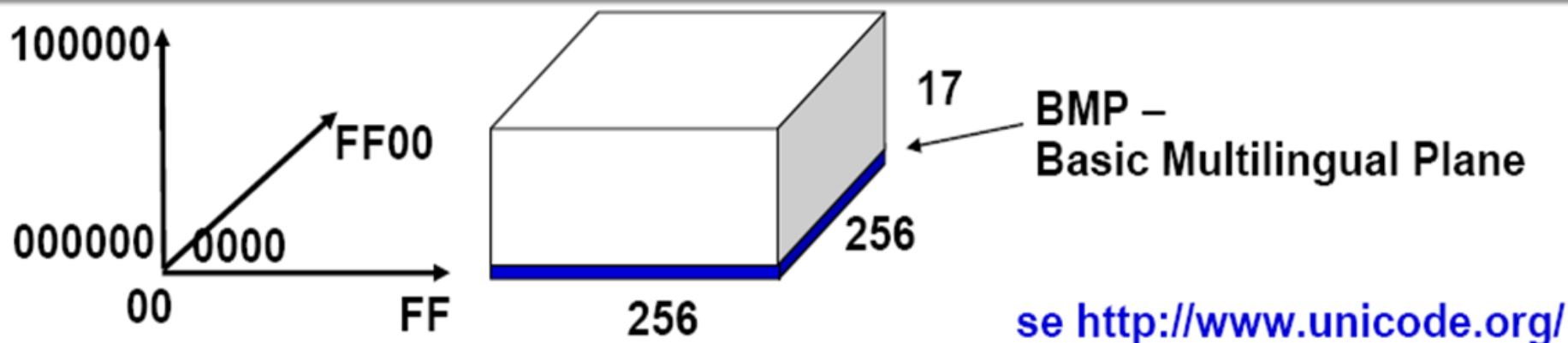
- Forsøk på å løse tegn-problemet en gang for alle
- Definerer bare **kode-punkter**, ikke **hvordan** de skal kodes binært
- 21 bit kodepunkt
- Faktisk **binær kode** bestemmes av **Transformasjonsformatet**
  - UTF-32, UTF-16, **UTF-8**, UCS-2, UTF-7 m.fl.
- Kodepunkter noteres typisk hexadesimalt, med U+ som prefix
  - U+0041 er kodepunktet for A

# Unicode STANDARDEN

---

- Er ikke et tegnsett, men definerer koder for elementer som kan være med ("codespace").
- Sier ikke noe om HVORDAN disse tegnene skal kodes binært.
  - Flere muligheter: UTF-32 (hele), UTF-16 (deler), UTF-8 (hele+), Windows-1252 (litt), ISO-8859-1 (litt)
- Kodene U+0000 til U+007F er lik ASCII (7bit)
- Kodene U+0080 til U+00FF er lik ISO 8859-1
- Inkludere alle kjente tegn + Tengwar o.l.

# UNICODE Struktur



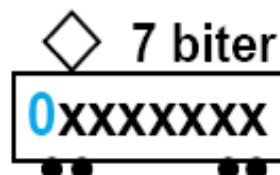
- Bruker (foreløpig) **21 bit** til kodepunkter
- Delt i 17 plan på 65536 kodepunkter i hvert
  - Plan 0: Basic Multilingual Plane (BMP)
    - Alle tegn vi støter på i «vanlige» språk
  - Plan 1: SMP
    - Historiske språk (f.eks hieroglyfer), musikk (noter), emotikoner ☺, spillkort mm
  - Plan 2: «Uvanlige» kinesiske tegn
  - Plan 14: Spesialtegn for tagging av språk o.l.
  - Plan 15-16: Privat bruk
    - Fimalogoer o.l. kan registereres her.

# Kodepunkter og transformasjoner

- Hvordan kodepunktene faktisk skal kodes er opp til bruker.
- Mest brukt er UTF-8 og UCS-2 (subset av UTF-16)
- **UTF-16**
  - Blir for alle («våre vestlige») praktiske formål lik Unicode kodepunktet
  - UCS-2 er alltid to byte (brukes primært på Windows)
  - *UTF-16 kan* være mer enn to byte, er det stort sett ikke
  - Kodingsformatet for String i Java
- **UTF-8**
  - U+0000 – U+007F er som i ASCII
  - Der etter benyttes to (eller flere) byte til å kode et tegn

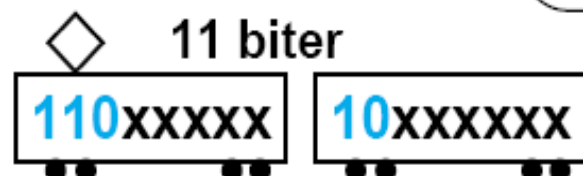
# Unicode UTF-8

- Enslig motorvogn = 0  
+ ASCII-kode

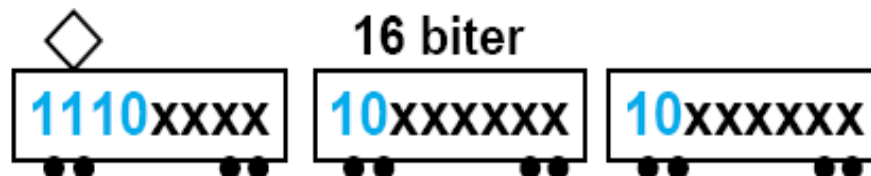


ASCII med en ekstra  
ledende 0  
er kompatibel med  
UTF-8

- Motorvogn i tog  
begynner alltid med et  
antall 1er-biter  
etterfulgt av en 0



- Antall 1er-biter i  
motorvognen  
= antall vogner i toget



- Vognene begynner  
alltid med 10

- Disse bitmønstrene  
brukes ikke for  
vanlige tegn i UTF-8



# Eksempel «Å» i UTF-8

- Tegnet «Å» har Unicode-punkt: U+00**C5**
- Binært: 0000 0000 **1100** **0101**
- I UTF-8 må vi da legge dette inn i **to byte**:

Mest signifikante (MSB)  
starter med 110

Minst signifikante (LSB)  
starter med 10

**110**x xxxx

**10**xx xxxx

**110**0 00**11**

**10**00 **0101**

«Padding»

- **UTF-8** koding av U+00C5 er **C385** ...



# Byte Order Marker

- Siden ett tegn kan kodes med mer enn en byte kan problem med *Endianness* oppstå
- Unicode-standarden innførte derfor **BOM**
  - Kode på starten av filen som viser hva slags rekkefølge byte legges inn i.
  - Mange moderne tekst-editorer legger automatisk inn BOM
- UTF-16 BOM: **U+FEFF**
  - **Little Endian (UTF-16LE)**: 0xFF fulgt av 0xFE (**p**ÿ)
  - **Big Endian (UTF-16BE)**: 0xFE fulgt av 0xFF (ÿ**p**)
- UTF-8 BOM: **U+FEFF**
  - Blir da 0xEF 0xBB 0xBF (ï»¿ som ISO-8859-1)
  - Leder lett til problemer, er ikke krevet av standarden og trengs egentlig ikke da innledende bits identifiserer plassering entydig.
  - Java støtter ikke BOM i UTF-8

# Experiment i Word (Windows)

---

Setningen

Æ kodes som 00C6

blir når benytter Alt-X på hvert enkelt tegn i Word:

00C60020006B006F00640065007300200073006F006D00200030003000430036

# Windows programmering

---

- Alle Windows C APler som omhandler tekst har 2 versjoner; A og W
- CreateFileA() dokumentert som ANSI
  - Støtter i utgangspunktet 1252 ASCII, men erfaring tilsier at å bruke annet enn 7-bit ASCII er å be om trøbbel
- CreateFileW() dokumentert som Unicode
  - Er egentlig UCS-2, vil alltid være 2 byte, lagres som W\_CHAR i usermode (16 bit), og UNICODE\_STRING i kernelmode



# BILDER OG LYD

- Kan kun bli en smakebit i dag
- Bilder
  - Finnes flere hundre ulike formater
  - skiller mellom
    - **bitmap**/raster og **vektor** formater
    - **Komprimering**: tapsfri eller ikke?
- Lyd
  - Må samples for å digitaliseres
    - **Nyquists** samplingteorm
- Film
  - Container-format



# FILFORMATER

- En fil er (vanligvis) en **bitstrøm** som har blitt gitt et **navn**
- De aller fleste filformater består av
  1. meta-data og
  2. data
- Metadata beskriver hvordan filen skal behandles av programmet som skal bruke filen
- Metadata ligger vanligvis i en **HEADER**

## Header

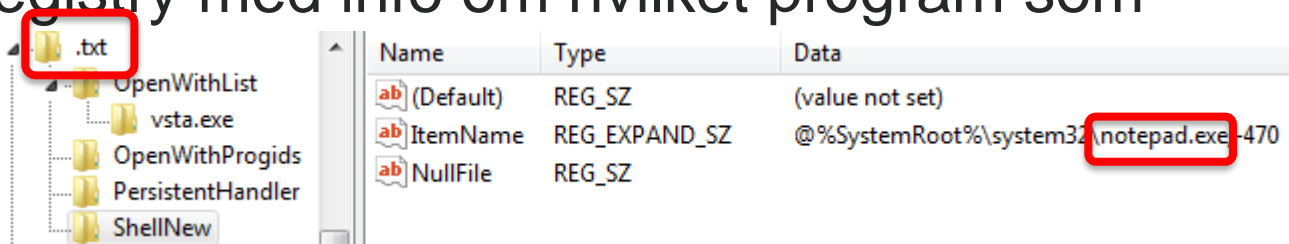
- beskriver dataene i filen

## BODY

- Data vi skal bruke til noe

# «Magic Number»

- Mange filformater starter med et «magisk tall» som fungerer som ID for filformatet
  - Windows bruker også fil-endelser som IDer og legger dem inn i Registry med info om hvilket program som åpner dem.



- **OSX** bruker et mer «distribuert» system
  - plister som kan endres med (bl.a.) `defaults` kommandoen
- Noen eksempler på magiske tall i filformat

- Java **.class** filer:

```
0001 0203 0405 0607 0809 0A0B 0C0D 0E0F 0123
0x00 CAFE BABE 0000 0032 003D 0A00 1300 1C09 Êp°¾
```

- Windows PE .exe filer:

0x00 4D5A 9 MZ

- Adobes PDF filer:

0001 0203 0405 0607 0809 0A0B 0C0D 0E0F 012345  
0x00 2550 4446 2D31 3334 0D25 E2E3 CFD3 0D0A %PDF-1

- MS Office (gammelt):

0001 0203	0123
0x00 <b>D0CF 11E0</b>	Di.à

# Binære vs text-filer

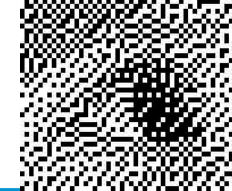
- **Text-filer** inneholder alltid binære koder som skal tolkes som text
  - «Alle» skjønner formatet
    - «selvforklarende»
  - Enkle å overføre mellom ulike plattformer
    - interoperabilitet
  - Kildekode, **XML**, HTML, CSS, SVG,..
  - Kan vanligvis gå ut fra at de består av byte.
- **Binære filer** inneholder binære koder som skal tolkes i henhold til et gitt format
  - Må vite filtype for å kunne tolke den korrekt
  - Kompilerte programmer, .class, .exe, .dll, .bmp, .pdf, .doc, ...
  - Kan ikke gå ut fra at de respektere noen faste **byte-grenser**.

---

# BILDER OG LYD



# Representasjon av bilder



- Bildet deles opp i små elementer (**pixel**) som får en verdi

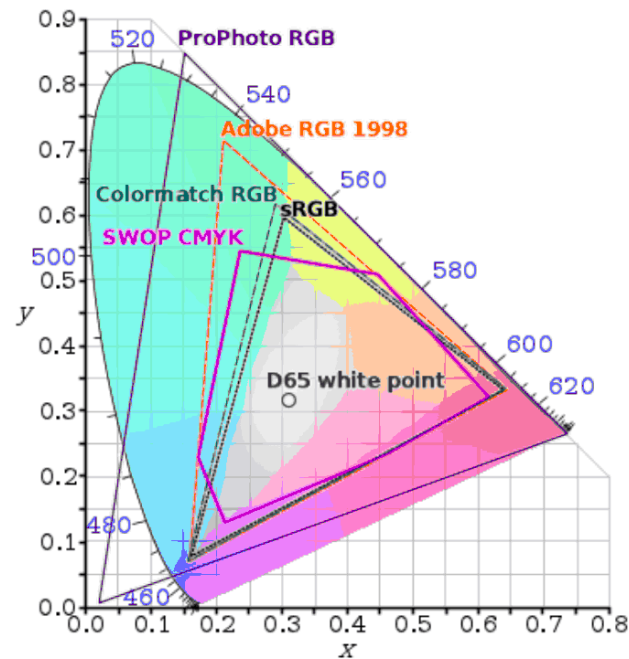
```
010101010101010101010110101101001001000111110000
01101010101010101010101001011010010110010100000110
100101010101010101010110110001010000101001010100
101101101011011010110101100110010110100010001001
011010010110100101101010001001100100101101010010
100101101100101011010101110110011001010010101100
011010010011010110010010001001100110101010010001
010101101100101100100101110110011001010100100101
010101010101010011011010001001100010100001010100
101010101010101100010010110010001101001110100001
010101010101010001000101000101101000010000001101
110110101010010100110100011010010011100101101000
101001010100100010100101100101101100001010000010
1010110100010010010010010111101011010100101100
10101000010001001001011111010111100101001001001
0101001010010001001010101110101011010010010000
1010010000100110011011111010111010101000100101
01001001010010001101100001110111011010110101000
00010000000100110010011111111110110111000000010
101000101010010011011000010101011101000010101000
00001000010010110101001111111111111011101000101
001000101001101010100100011101111110100010010000
010010010110001001001001111011110101101100100101
100100100000111010010010010111111111011001001000
```



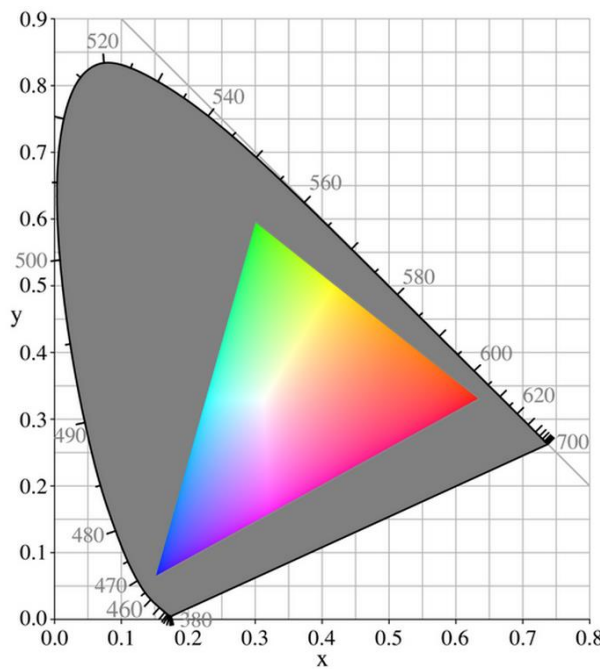
**MER OM DETTE I DAGENS ØVING  
T02.02!**

# Farger: sRGB

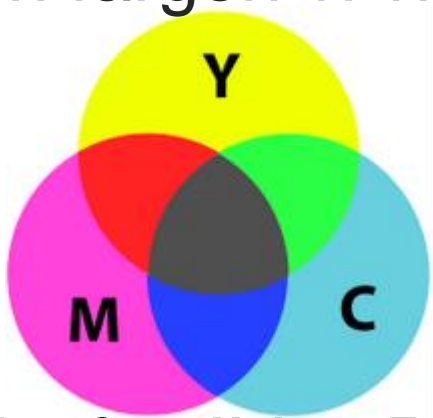
- sRGB er standard farge-rom på Internett
  - Spesialtilpasset egenskapene til en CRT-monitor
  - En del nyanser lar seg ikke representere.
  - Typisk 1 byte hver for R, G og B; og gjerne en 1 byte for gjennomskinnlighet (alfa-kanal)



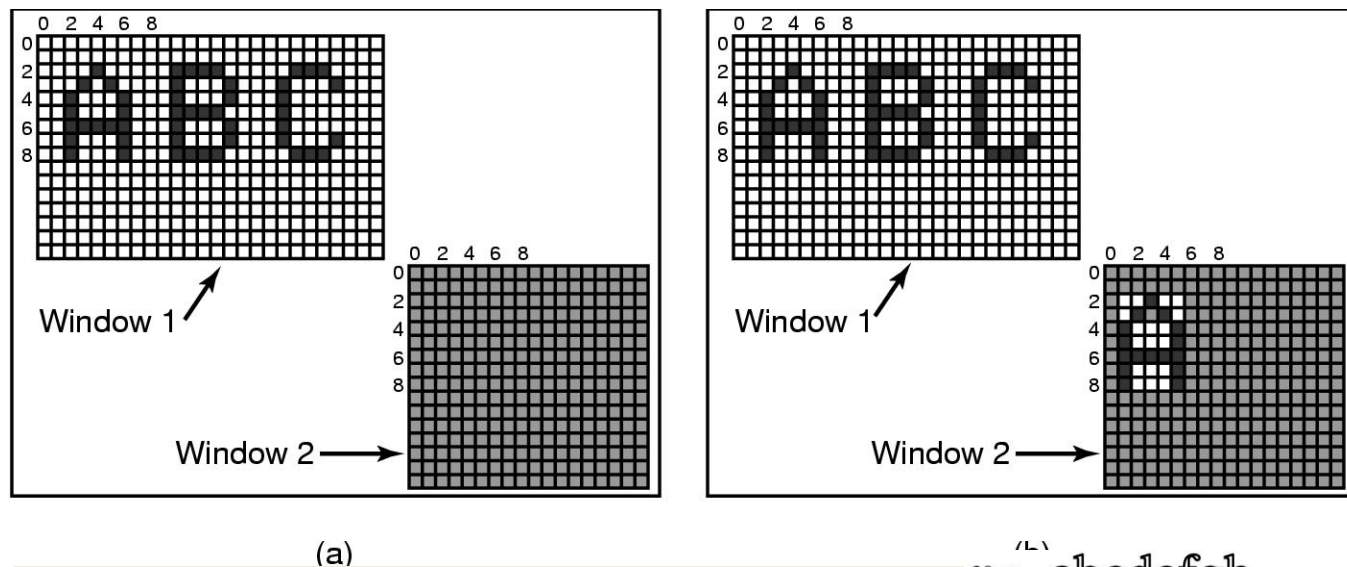
"CIE1931xy gamut comparison" by BenRG and cmglee  
- [http://commons.wikimedia.org/wiki/File:CIE1931xy\\_blank.svg](http://commons.wikimedia.org/wiki/File:CIE1931xy_blank.svg).



# Utskrift: CMYK

- Farger fra skjerm er *additative*
    - Skjermen stråler ut den fargen vi vil se.
  - Farger på utskrifter er *subtraktive*
    - «blekket» reflekterer den fargen vi vil se og absorberer de andre.
- 
- A Venn diagram illustrating the subtractive color model CMYK. It consists of three overlapping circles labeled Y (Yellow), M (Magenta), and C (Cyan). The intersections of two colors create secondary colors: Y and M overlap to form Red, Y and C overlap to form Green, and M and C overlap to form Blue. The central intersection where all three colors (Y, M, and C) overlap is labeled K (Black), representing the result of subtracting all primary colors from white light.
- Printerer o.l. benytter derfor ikke RGB, men ofte et CMYK-fargerom
    - K er sort, da blandingen ikke er intens nok...

# Bitmap vs vektorgrafikk



**Bitmap:** koder og lagrer enkelt-pixlene slik de skal vises på skjermen

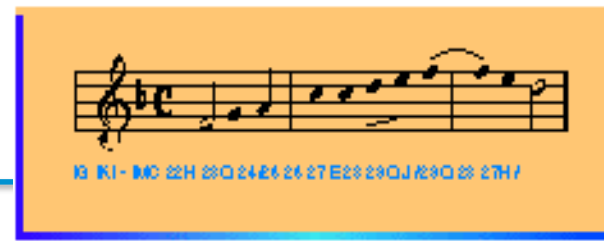
**Vektor:** Finner matematisk formel (ofte **Bézier** kurver) for å lage formen på billed-elementene du vil ha og lagrer bare formelen

20 pt: *abcdefgh*

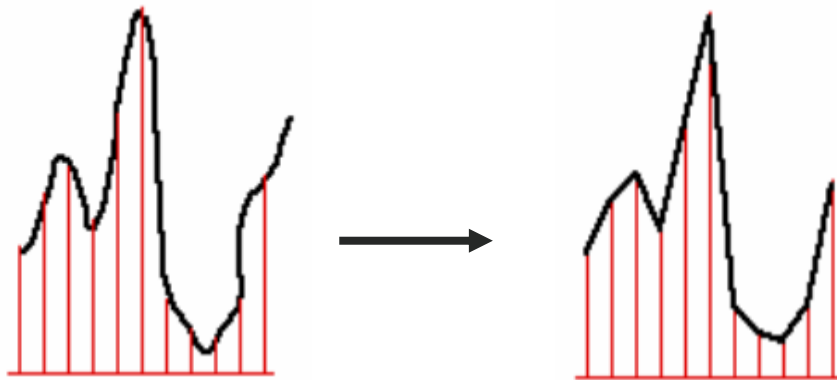
53 pt: *abcdefgh*

81 pt: *abcdefgh*

# Representasjon av lyd



- Lyden blir digitalisert (samplet)
  - Hver liten lydbit blir representert av frekvens



# Lyd

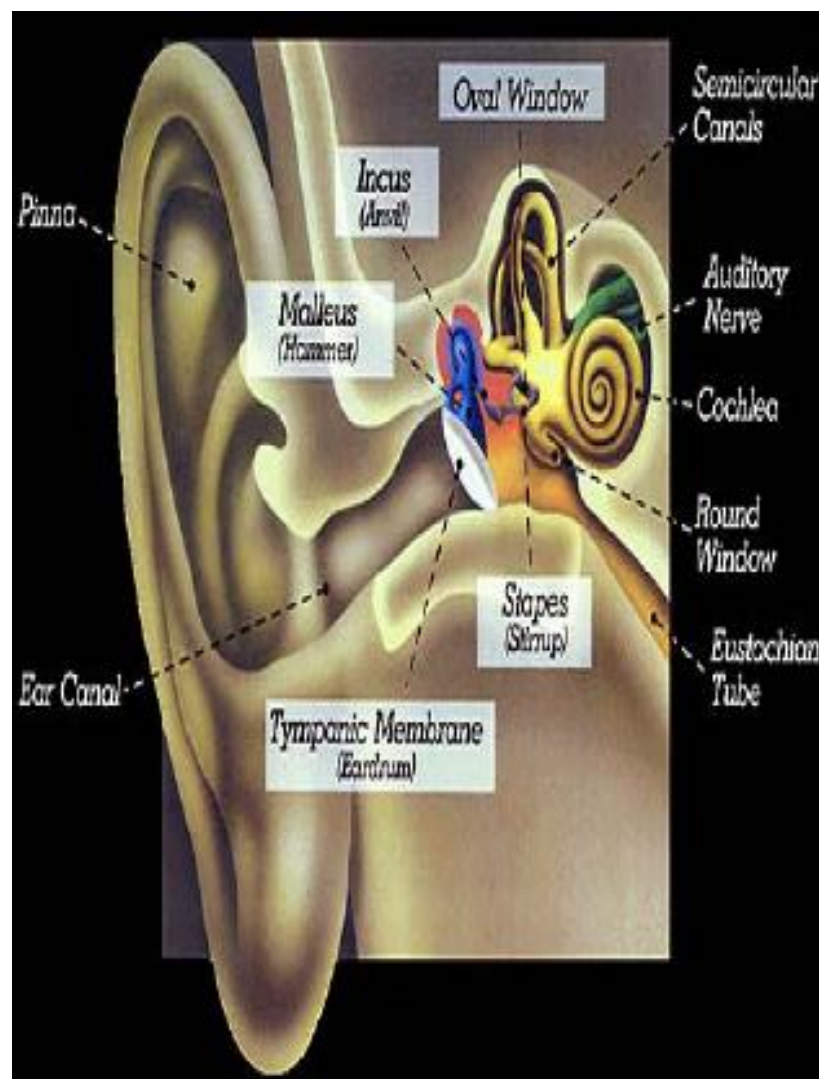
- Med lydkort i maskinen kan den lage multimedia presentasjoner
- Lyd kan lagres digitalt (f.eks. MP3)
- Tale kan digitaliseres og avspilles syntetisk
- Musikk kan lages med synthesizer (MIDI)



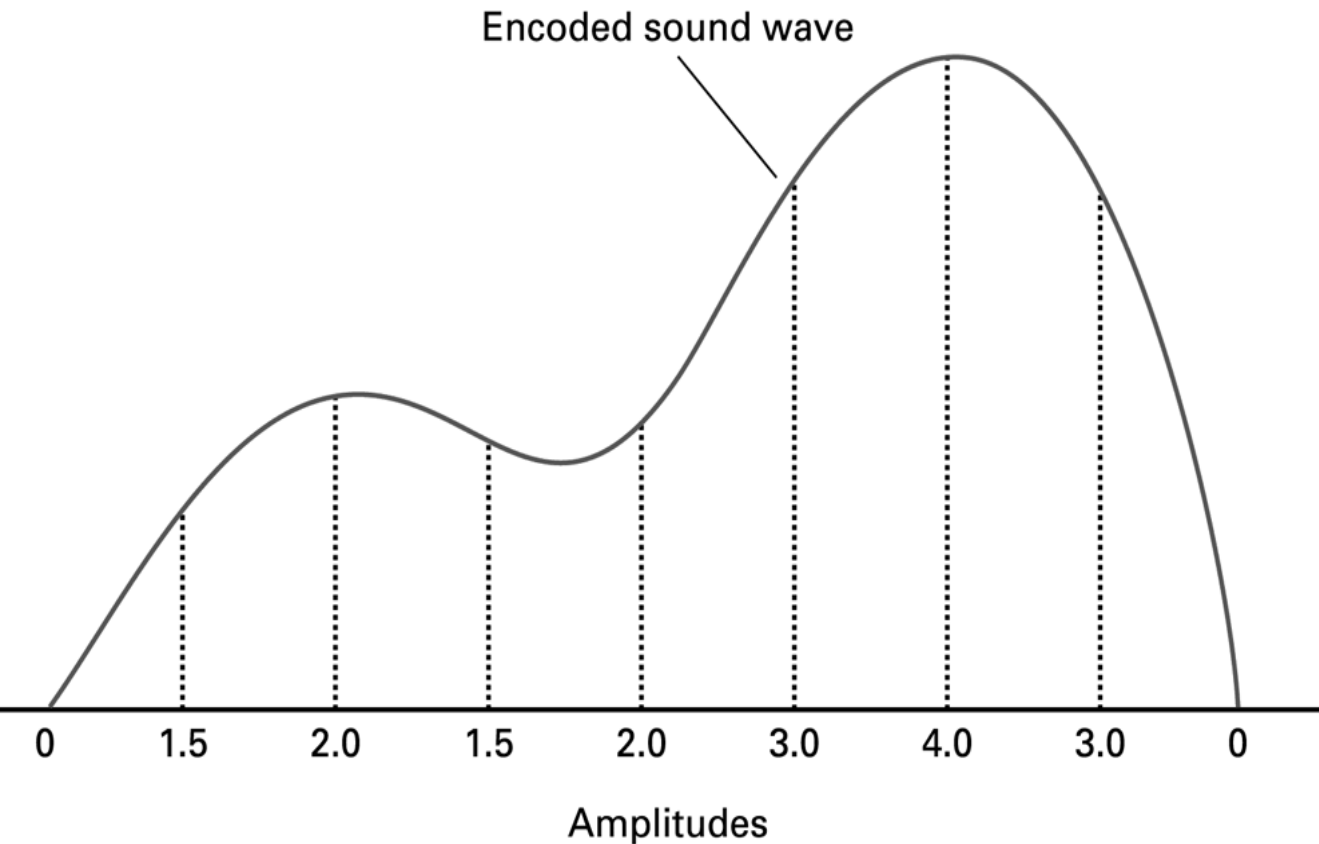


# Øret

- Lydbølger detekteres av øret, omformes til nerveimpulser, og disse sendes til høresenteret i hjernen.
- Det ytre øret øker sensitiviteten med en faktor 2 – 3.
- Resonans i øregangen øker følsomheten for lyder ved 3 000 – 4 000 Hz. (viktig for oppfattelse av tale)
- Tromhinnen forsterker ca 15x
- I det indre øret sitter 16 000 – 20 000 hårceller som registrerer lyd med forskjellige frekvenser.
- Vibrasjonene fra trykkbølger bøyer hårene, ionekanaler i bunnen av cellene åpnes, og en liten strømpuls sendes langs en nervefiber til hørselsenteret i hjernen.



# Lyd - digitalisering

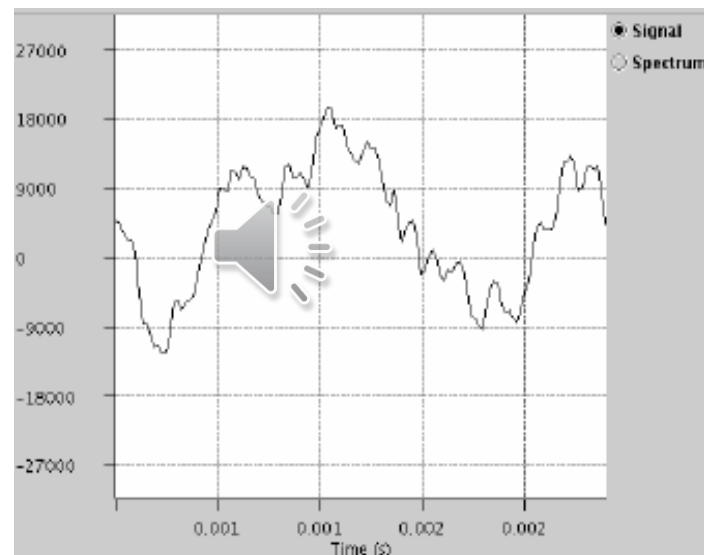
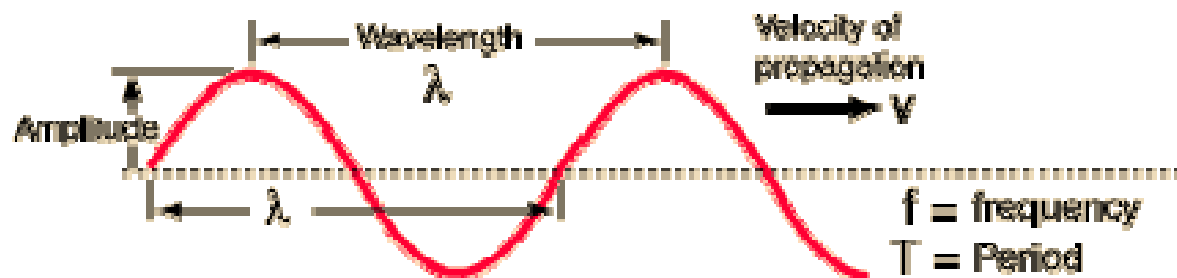


Digitaliseres til:  
0, 1.5, 2.0,  
1.5, 2.0, 3.0,  
4.0, 3.0, 0  
(sampling)

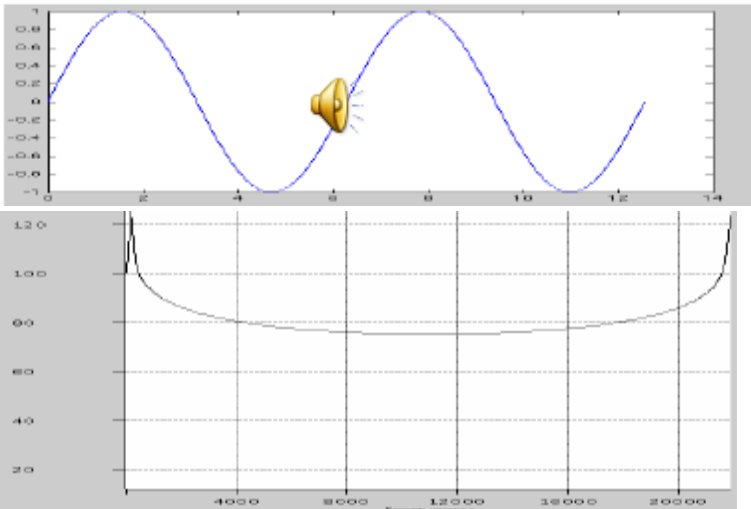


# Lydbølger

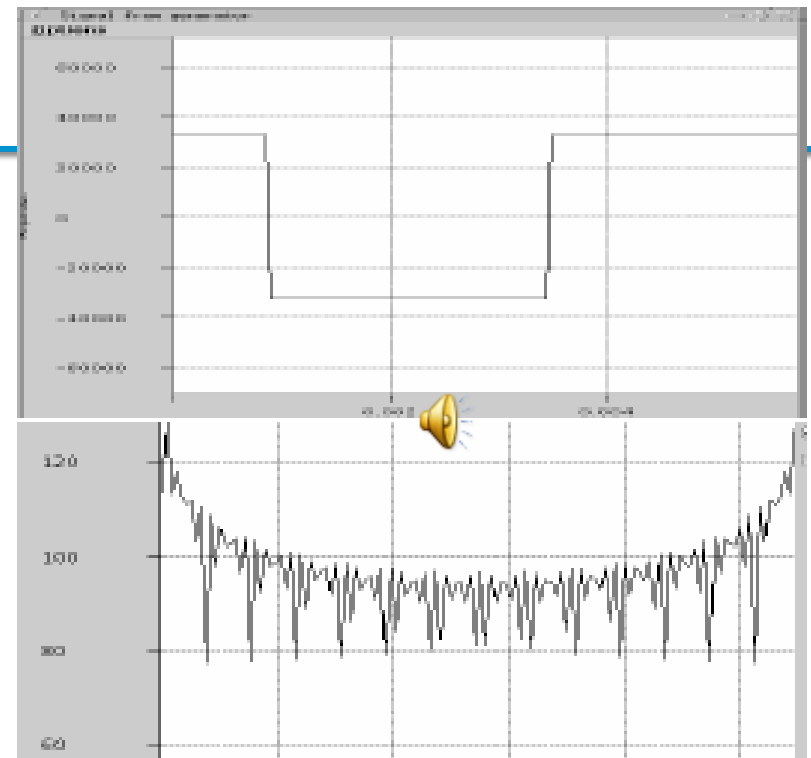
- **Nyquist-teoremet**
- Vi må **sample amplituden** med en **samplingfrekvens**, som for å få (perfekt) gjengivelse må være minst **dobbel** så stor som en **største frekvenskomponenten** i lyden vi skal digitalisere



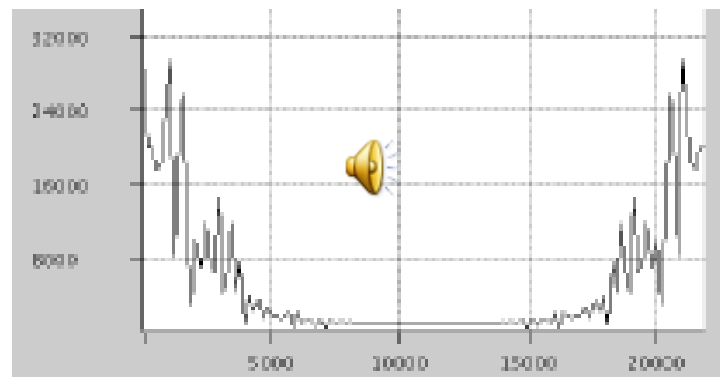
# Frekvensspekter



Sinus

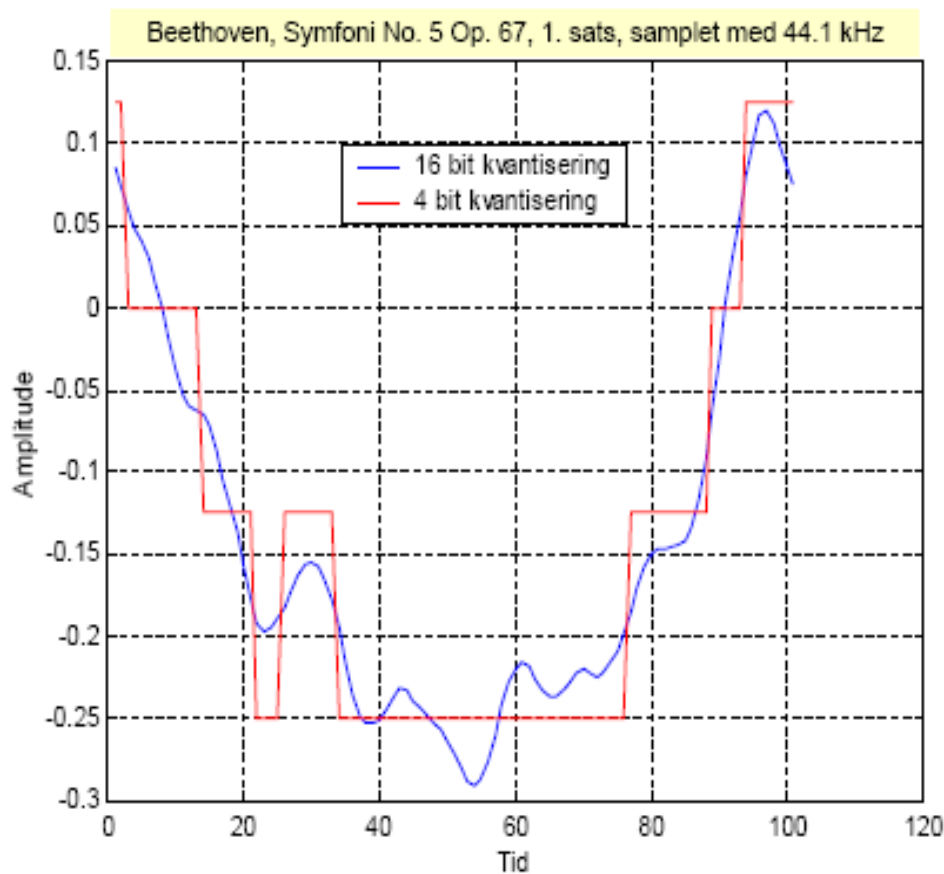




Firkantpuls



Messingblåser

# Kvantisering eksempel



- Beethovens 5. kvantisert med hhv 16 (CD) og 4 bit (halvparten av telefon), samme samplingfrekvens
- 16 bit-> 
- 4 bit -> 

---

# KOMPRIMERING

# Data-komprimering

- Ved hjelp av mønster-gjenkjenning kan datamengder som skal lagres gjøres mindre
- Eksempel:
  - En hest fra vest er best i test for de fleste (45 tegn)
  - En h<sup>1</sup> fra v<sup>1</sup> er b<sup>1</sup> i t<sup>1</sup> for de fl<sup>1</sup>e      <sup>1</sup> = est (35 tegn)
- Det må benyttes egne programmer både for å komprimere data og pakke disse ut igjen
- Grafikk og multimedia kan ofte bli sterkt komprimert
- Populære formater er
  - gif, png, jpg, mpg,
  - zip, arj, .....

# Typer komprimering

- **Tapsfri** versus med tap
- **Run-length** encoding
  - Lange, like sekvenser erstattes med en kode (f.eks. 583 0'er erstattes med 583\*0)
- **Frekvensavhengig** koding (Huffman codes)
  - De vanligste kodene får kortest representasjon (jf Morse)
- Relativ/**diffrensiell** koding
  - Koder forskjell mellom påfølgende enheter istedetfor hver enkelt enhet (eksempel: stemme over GSM)
- **Ordbok**-koding
  - LZW (jf neste foil, er adaptiv ordbok-koding: lager ordboken etterhvert som man koder meldingen)

# Lempel, Ziv, Welch

- Lager ordboken mens vi koder (og dekoder).  
Exempel:
  1. xyx xyx xyx xyx -> ordbok: x=0, y=1, mellomrom = 2
  2. Koder: 0102, mellomrom viser at xyx er et ord => ordbok: xyx = 3
  3. 0102333 blir kodingen: komprimering  $7/15 = 47\%$

NB! Ordboken *følger av* rekkefølgen, og trenger ikke følge med de komprimerte dataene!! Den kan lages på nytt med grunnlag i selve den omkodede meldingen

- Dekomprimering: 0102 -> xyx mellomrom -> xyx = 3, osv.

SE D02.00 i ITL



# Komprimere bilder

---

## GIF

- Ordbokkoding, 256 farger pr pixel som lagres i ordbok, tap av fargedybde, en farge er "gjennomsiktig", egner seg for enkel animasjoner.

## JPG

- Mange forskjellige komprimeringsteknikker, utnytter at øyet er mer følsomt for endring i lysstyrke enn farge, "baseline standard" komprimerer 10-30 ganger, egner seg for foto.

## PNG

- Open source versjon av JPG, bruk = web

## TIFF

- Komprimering minner om GIF, men brukes mye pga evne til å legge inn mye metadata, til arkivering



# Komprimere lyd og fil: MPEG

---

- Film
  - Benytter differensiell koding, velger noen bilder (I-frames) som komprimeres i sin helhet (teknikk a la JPG), og *koder bare forskjell* for resten.
- Lyd (MP3 – MPEG Layer 3)
  - Fjerner detaljer som øret ikke hører, f.eks.
    - hører man i en periode *etter* en høy (intens) lyd ikke svakere lyder (tidsmaskering).
    - Nærliggende frekvenser med lavere intensitet blir "overdøvet" av de med høyere intensitet (frekvensmaskering)

# Multimedia Container-formater

- De fleste **multimedia**-formater er egentlig **container/wrapper-formater**
  - De beskriver *hvordan* bilder og lyd som er pakket inn i formatet er kodet
  - Inneholder flere **separate data-strømmer**
- **Codec (coder-decoder)**
  - Programvaren som foretar den faktiske omkodingen til skjerm og høyttaler
- Eksempel **MP-4** (MPEG-4 Part 14)
  - Videreutvikling av Quicktime-formatet
  - Kan innholde data-strømmer for
    - Metadata i MPX
    - Undertekster: flere formater
    - Film og lyd: Se f.eks. <http://www.mp4ra.org/codecs.html>
- At en fil faktisk lar seg avspille avhenger dermed at du har de korrekte codec'ene, ikke bare avspiller-programmet

---

# DAGENS ØYINGER

# Hexeditor

- I dagens øving skal vi bruke en HexEditor for å utforske oppbyggingen av f.eks. et BMP-bilde..
  - En **Hexeditor** er programvare som viser deg og lar deg endre det binære innholdet i en fil
  - For å løse oppgaven **må** man samtidig følge med i et dokument som ligger i samme mappe som oppgaven og som beskriver den binære kodingen av bildet!
- **Windows** brukere kan f.ex. laste ned og bruke TinyHexer
  - [http://texteditors.org/cgi-bin/wiki.pl?Tiny\\_Hexer](http://texteditors.org/cgi-bin/wiki.pl?Tiny_Hexer)
  - Det finnes også en HexEd-PlugIn til Notepad++
- **OSX** brukere kan f.eks. bruk 0xED
- **Linux**-brukere kan f.ex. bruke DHEX

# HexEditor (2)

Binært innhold i filen  
Vist hexadesimalt

Innhold i filen  
Vist som ASCII

	0001	0203	0405	0607	0809	0A0B	0C0D	0E0F	0123456789ABCDEF
0x00	424D	8E98	0000	0000	0000	3600	0000	2800	BMŽ~.
0x10	0000	6300	0000	8200	0000	0100	1800	0000	..c..
0x20	0000	0000	0000	130B	0000	130B	0000	0000	.....
0x30	0000	0000	0000	0000	0000	0000	0000	0000	.....
0x40	0000	0000	0000	0000	0000	0000	0000	0000	.....
0x50	0000	0000	0000	0000	0000	0000	0000	0000	.....
0x60	0C							000	.....
0x70	0C							000	.....
0x80	0C							000	.....
0x90	0C							300	.....
0xA0	0003	0000	0100	0100	0402	0002	0200	0000	.....
0xB0	0100	0511	293E	5A16	2058	0F2C	5911	2B5A	.....):
0xC0	192E	5B15	2A57	102C	5B13	3465	1F43	7320	..[.*W.,[.4e.Cs
0xD0	4878	1B47	761D	4E7C	2357	8521	5A87	2058	Hx.Gv.N #W...!Z† X
0xE0	871C	5785	1C57	851D	5886	2157	861E	5483	†.W...W...X†!W†.Tf
0xF0	1F53	821E	5483	2259	8C23	5C90	245C	9326	.S,.Tf"Y&#\ \$\"&
0x0100	6199	266C	A12A	72A2	3476	9F4C	7FA0	3147	a™&l;*r¢4vŸL 1G
0x0110	6000	000D	0502	0B08	0304	0002	0201	0402	`.....

På posisjon **0xC8** er det en byte med verdi **0x5B**  
= 0101 1011 = 91<sub>10</sub> , som ASCII er det glyfen [



# Instruksjoner til datamaskinen

- Instruksjoner til datamaskinen = programmering

## HØY-NIVÅ

```

INPUT A
INPUT B
FOR I = A TO 0 STEP -1
SUM = SUM + B
NEXT I
OUTPUT SUM
END
    
```

## ASSEMBLER

```

START:  IN      A
        IN      B
        STORE   B
        LOAD    A
        SUB     B
        FOR     0
        ADD     SUM
        NEXT    0
        BRANCH  FERGIG:
        LOAD    SUM
        OUT
        STOP
FERGIG: DM      0001
SUM:    DM      0000
A:      DM      0000
B:      DM      0000
    
```

## BINÄRT

```

0000 1001 0000 0000
0000 0010 0001 0000
0000 1001 0000 0000
0000 0001 0001 0010
0000 0001 0001 0001
0000 0100 0000 1111
0000 0001 0001 0001
0000 0000 0000 1100
0000 0001 0001 0010
0000 0011 0001 0000
0000 0000 0001 0000
0000 0001 0000 0100
0000 0001 0001 0000
0000 1010 0000 0000
0000 0000 0000 0000
0000 0000 0000 0001
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
    
```

## HEX

```

0900
0210
0900
0112
0111
040F
0211
070C
0112
0310
0210
0504
0110
0400
0000
0001
0000
0000
0000
0000
    
```

NESTE GANG!!!