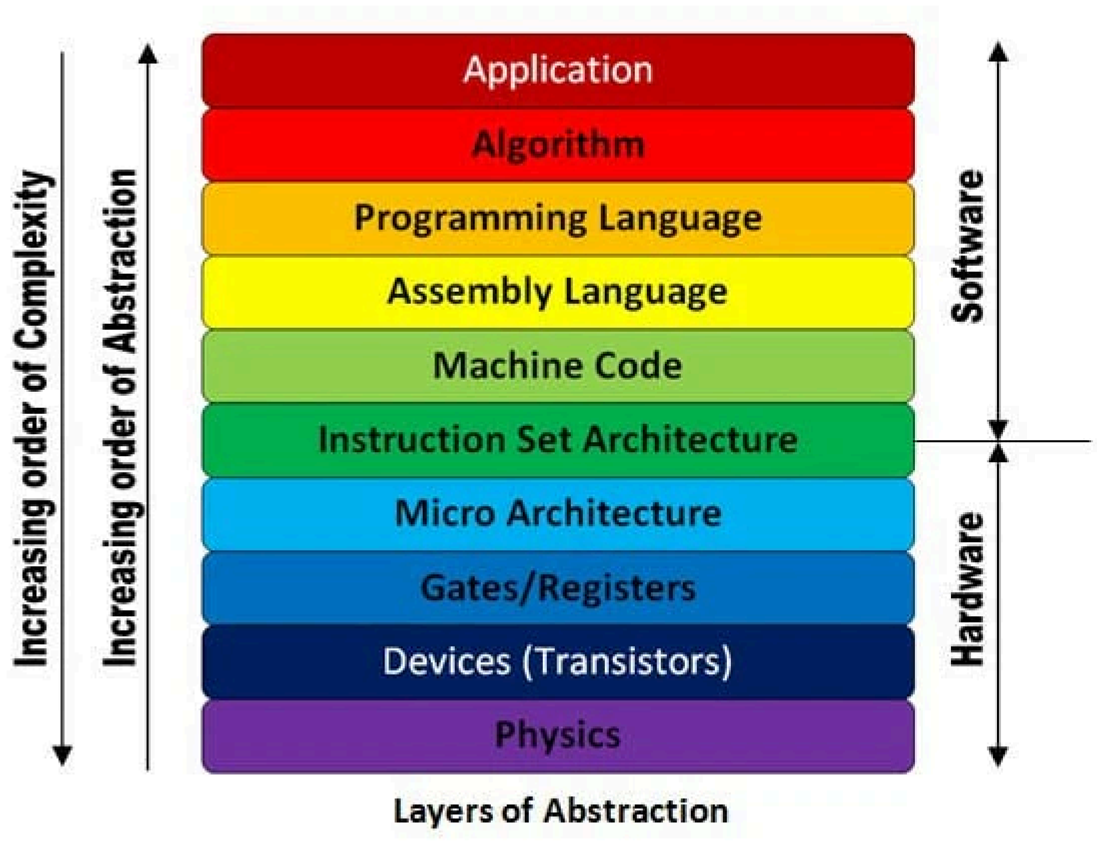
Music and Computation

Stephen Slade
Yale Computer Science Department

Video - Title of the Song https://www.youtube.com/watch?v=734wnHnnNR4
DaVinci's Notebook

Abstraction

A fundamental principle of Computer Science is **abstraction** - creating artifacts from primitive elements and successive layers of high level constructs. For example, a typical digital computer might comprise the following successive layers of hardware and software:



https://softwareg.com.au/blogs/computer-hardware/hardware-layer-in-computer-architecture

Numbers

Spoiler alert: everything in the computer boils down to zeros and ones – binary numbers. A single zero or one is a **binary digit** or **bit**. All digital data comprises a string of bits: numbers, text, images, sounds, video - everything.

Moreover, the fundamental theoretical definition of computation is simply a process that converts one string of bits to another string of bits. The Turing Machine is the epitome of this model. See https://en.wikipedia.org/wiki/Turing_machine

We start with binary numbers - base 2. Here are some examples of positive integers in binary and decimal:

Binary Number (base 2)	Decimal Number (base 10)
1	1
10	2
100	4
101	5
1010	10
100001	33
1000001	65
1100100	100

Though computers excel at binary or true/false logical representations, most people find it difficult to process binary numbers. We can employ base 8 or **octal** to group binary digits. The octal digits range from 0 to 7. The octal number 10 represents 8. See https://en.wikipedia.org/wiki/Octal Below we add a column of octal to the previous table.

Binary Number (base 2)	Decimal Number (base 10)	Octal Number (base 8)
1	1	1
10	2	2
100	4	4

101	5	5
1010	10	12
100001	33	41
1000001	65	101
1100100	100	144

We can go one step up by a power of 2 to base 16 or hexadecimal. In this case, we need additional digits for the numbers 10, 11, 12, 13, 14, and 15. We use the initial letters of the alphabet, namely, A, B, C, D, E, and F - either upper or lower case. See https://en.wikipedia.org/wiki/Hexadecimal Below we add another column of hexadecimal to the previous table.

Binary Number (base 2)	**Decimal Number (base 10)**	**Octal Number (base 8)**	**Hexadecimal Number (base 16**
1	1	1	1
10	2	2	2
100	4	4	4
101	5	5	5
1010	10	12	A
100001	33	41	21
1000001	65	101	41
1100100	100	144	64

It is important to note that converting an integer from binary to decimal to octal to hexadecimal is exact. There is no loss of information. You can go back and forth among the bases with no rounding errors or loss of information. It is a lossless process. See https://en.wikipedia.org/wiki/Lossless_compression We can view the conversion of the binary number 1100100 to the hexadecimal number 64 as a lossless compression. We went from 7 digits to 2 digits.

We note that negative integers are usually represented by a sign-bit prefix to the binary representation. See https://en.wikipedia.org/wiki/Signed_number_representations Again, this is a lossless process.

There are limits of precision to integer representations largely due to the finite size of computer words. Common word sizes are 32 bits and 64 bits. A 32 bit computer, with an initial sign bit, could not represent an integer greater than 2^{31} or 2,147,483,648. A computation that exceeded that value would cause an integer overflow error. See https://en.wikipedia.org/wiki/Integer_overflow Similarly, a 64 bit computer with a sign bit would have an upper limit of 2^{63} or 9,223,372,036,854,775,808. For most applications, you are not likely to exceed either limit, for example, in calculating your income taxes.

There are many other flavors of numbers, including rationals, reals, decimals, floating point, complex, imaginary, and irrational. Some languages, like racket, have exact representations for rationals, like ⅓, which is represented by the integers 1 and 3. However, many languages will convert rationals to their decimal approximations, such as .3333. Note that .3333 does not equal ⅓. There is a rounding error. In racket, .3333 is an inexact number, while ⅓ is an exact number.

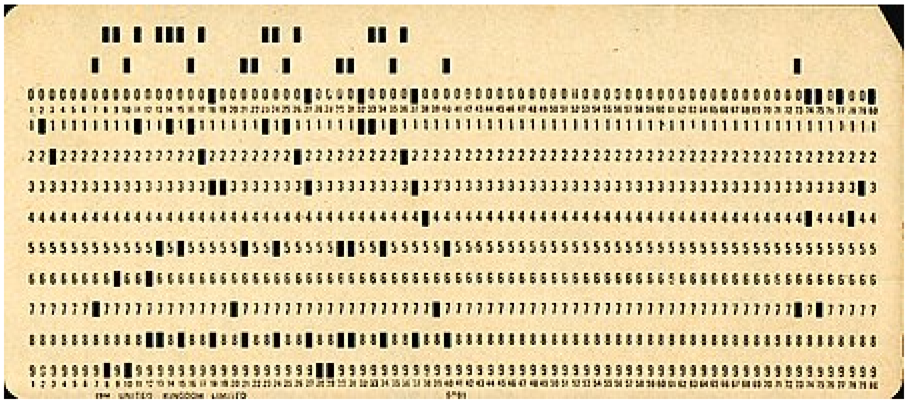
Decimal numbers are commonly represented using floating point notation. See https://en.wikipedia.org/wiki/Floating-point_arithmetic As in scientific notation, floating point numbers have a sign, a base, a mantissa, and an exponent. Floating points numbers are a subset of the real numbers, and often are inexact, that is, they have rounding errors. As such, conversion to floating point can be lossy. See https://en.wikipedia.org/wiki/Lossy_compression We will encounter lossy compression again in images, audio, and video.

Text

Text is ubiquitous in the digital world. Think of email, text messages, pdf documents, and web pages. The document you are reading now consists of digital text. Holy Recursion, Batman!

The digital representation of text is, wait for it, numbers. Each letter of the alphabet is encoded as a unique number, more precisely, a positive integer. In addition, the numerical encoding also imposes an order on the alphabet such that if you sort the text numerically, you also sort the text alphabetically, except for upper / lower case differences.

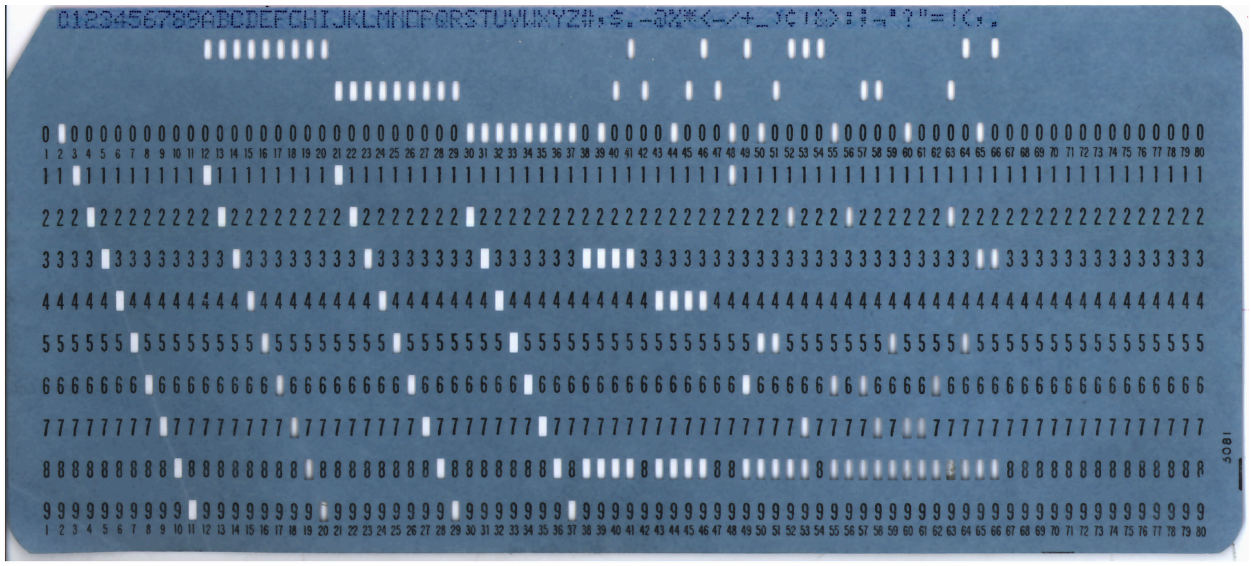
Before computers, there were punched cards. See https://en.wikipedia.org/wiki/Punched_card IBM dominated this technology. Below is a sample 80 column card.



Text on punched cards was represented by the patterns of holes punched in the card, left to right. See https://en.wikipedia.org/wiki/Punched_card This code is also known as Hollerith Code, for its inventor, Hermann Hollerith, who was the founder of the company that became IBM. See https://en.wikipedia.org/wiki/Herman_Hollerith The idea of machines controlled by punched cards goes back to 1804 and the Jacquard Loom, invented by Joseph Marie Jacquard. See https://en.wikipedia.org/wiki/Joseph_Marie_Jacquard

One of the earliest text encodings was Binary Coded Decimal or BCD, promulgated by IBM for punch card devices before the advent of computers. Also known as Binary Coded Decimal Interchange Code or BCDIC. See https://en.wikipedia.org/wiki/Binary-coded_decimalhttps://en.wikipedia.org/wiki/BCD_(character_encoding)

BCDIC was limited. For example, it had only upper case letters. IBM extended the encoding to create EBCDIC (Extended Binary Coded Decimal Interchange Code) which was used in the IBM 360 family of computers which dominated the mainframe computer market starting in the 1960's. See https://en.wikipedia.org/wiki/EBCDIC


0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ#,$.-@%*<-/+_)¢|&>:;¬'?"=!(,.

An 80-column punched card with the extended character set introduced with EBCDIC in 1964.

ASCII and Unicode

The main alternative to EBCDIC was ASCII (American Standard Code for Information Interchange). See https://en.wikipedia.org/wiki/ASCII IBM was actually an advocate for ASCII, but was not able fully to support it in the IBM 360 rollout. However, when IBM brought out its personal computer or PC, it used ASCII. UNIX had already been in the ASCII camp.

BCDIC, EBCDIC, and ASCII shared a common failing. They were limited to the Roman or Latin alphabet required for English. An ASCII character is 7 bits, which means that you can have 2^7 or 128 different characters. That is sufficient for 10 decimal digits, upper and lower case letters, and a smattering of punctuation marks and control codes, like new line, tab, or carriage return. However, languages like French and German have diacritical marks, and other languages like Greek, Russian, Arabic, and Hebrew have non-Latin alphabets. Moreover, beyond languages like Chinese, Japanese, and Korean there are a boatload of other writing systems. See https://en.wikipedia.org/wiki/List_of_writing_systems

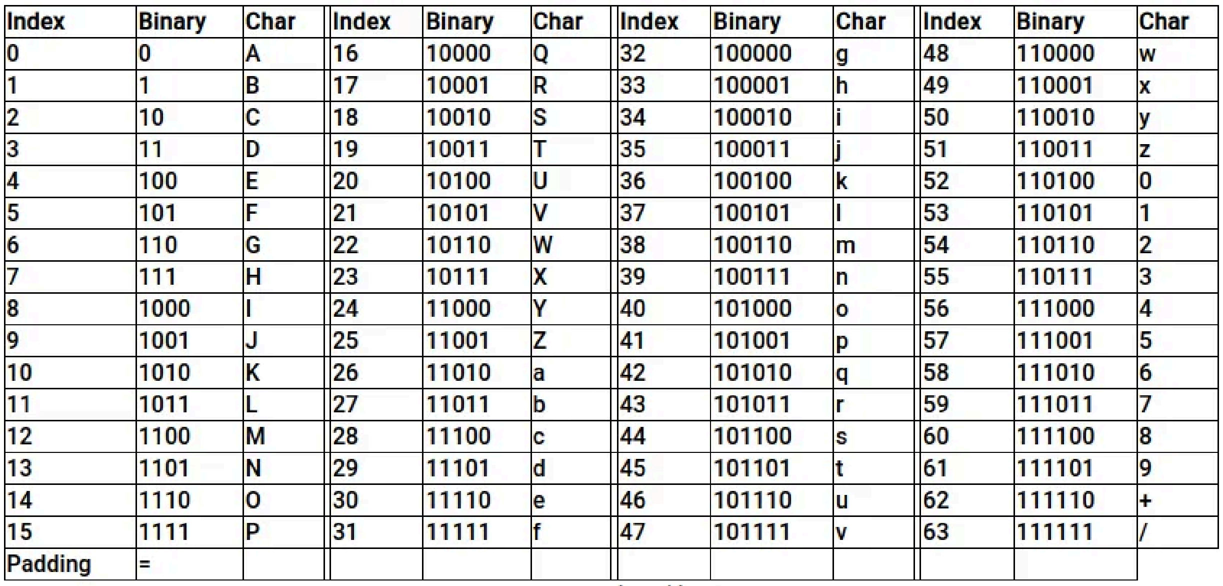
People can adapt to technology. It is possible to use the Latin alphabet to encode other languages. For example, there is a Yale system for encoding Mandarin Chinese. https://en.wikipedia.org/wiki/Yale_romanization_of_Mandarin

However, it is usually better for computers to adapt to people. Thus, in the 1980's researchers at Xerox and Apple developed Unicode, which currently comprises roughly 150,000 characters (or "code points") including emojis!
See https://en.wikipedia.org/wiki/Unicode

Unicode characters may be up to 4 bytes (32 bits) wide. The common UTF-8 format can include up to four bytes. See https://en.wikipedia.org/wiki/UTF-8

Base64

As you may know, bit strings can include non-graphic letters, including **whitespace** characters like newline and tab, as well as **control** characters, such as control-C or control-Z. Sometimes you want to be able to send a message or create a file in which the non-graphic letters appear as ink on the page (or pixels on the screen.) That's where Base64 comes in handy. See https://en.wikipedia.org/wiki/Base64 The 64 refers to the 64 characters which comprise the Base64 encoding. You need only 6 bits to represent 64 unique characters ($2^6 = 64$). Here is the translation table:



Index	Binary	Char	Index	Binary	Char	Index	Binary	Char	Index	Binary	Char
0	0	A	16	10000	Q	32	100000	g	48	110000	w
1	1	B	17	10001	R	33	100001	h	49	110001	x
2	10	C	18	10010	S	34	100010	i	50	110010	y
3	11	D	19	10011	T	35	100011	j	51	110011	z
4	100	E	20	10100	U	36	100100	k	52	110100	0
5	101	F	21	10101	V	37	100101	l	53	110101	1
6	110	G	22	10110	W	38	100110	m	54	110110	2
7	111	H	23	10111	X	39	100111	n	55	110111	3
8	1000	I	24	11000	Y	40	101000	o	56	111000	4
9	1001	J	25	11001	Z	41	101001	p	57	111001	5
10	1010	K	26	11010	a	42	101010	q	58	111010	6
11	1011	L	27	11011	b	43	101011	r	59	111011	7
12	1100	M	28	11100	c	44	101100	s	60	111100	8
13	1101	N	29	11101	d	45	101101	t	61	111101	9
14	1110	O	30	11110	e	46	101110	u	62	111110	+
15	1111	P	31	11111	f	47	101111	v	63	111111	/
Padding	=										

See https://www.debugpoint.com/bash-base64-encode-decode/

With Base64, you can convert **any** bit string into a set of printable characters! For example, you could take a compiled C program and make it text that you could include in an email. The idea is that you break the bit string into 6 bit numbers which are then replaced with the corresponding Base64 character.

Here is an example from the Wikipedia entry:

> Here is a well-known idiom from distributed computing:
>
> Many hands make light work.
>
> When the quote (without trailing whitespace) is encoded into Base64, it is represented as a byte sequence of 8-bit-padded ASCII characters encoded in MIME's Base64 scheme as follows (newlines and white spaces may be present anywhere but are to be ignored on decoding):
>
> `TWFueSBoYW5kcyBtYWtlIGxpZ2h0IHdvcmsu`
>
> In the above quote, the encoded value of *Man* is *TWFu*. Encoded in ASCII, the characters *M*, *a*, and *n* are stored as the byte values `77`, `97`, and `110`, which are the 8-bit binary values `01001101`, `01100001`, and `01101110`. These three values are joined together into a 24-bit string, producing `010011010110000101101110`. Groups of 6 bits (6 bits have a maximum of $2^6 = 64$ different binary values) are converted into individual numbers from start to end (in this case, there are four numbers in a 24-bit string), which are then converted into their corresponding Base64 character values.

There is a UNIX command, `base64`, which does the conversion.

```
$ echo Many hands make light work | base64
TWFueSBoYW5kcyBtYWtlIGxpZ2h0IHdvcmsK
```

The `-d` option will decode the text.

```
$ echo Many hands make light work | base64 | base64 -d
Many hands make light work
```

QR Codes

You are familiar with the ubiquitous quick-response or QR codes. See https://en.wikipedia.org/wiki/QR_code They are actually another way to encode text, without loss of information. Here is an example using the Python qrcode module

```
>>> import qrcode
>>> img = qrcode.make("https://zoo.cs.yale.edu/classes/cs201/index.html")
>>> img.save("qrcs201.png")
```

The following image is the result. Check it out with your phone.

The graphic layout was inspired by the board game go, with its alternating white and black stones.

While conventional bar codes are capable of storing a maximum of approximately 20 digits, QR Code is capable of handling several dozen to several hundred times more information.

QR Code is capable of handling all types of data, such as numeric and alphabetic characters, Kanji, Kana, Hiragana, symbols, binary, and control codes. Up to 7,089 characters can be encoded in one symbol. See https://www.qrcode.com/en/about/

What does the above QR code represent?

The bottom line: any digital text document of any size is simply a string of bits.

Images

Pictures, drawings, and photographs are commonly represented as dots on a screen or page. If the image is simply black and white, the dots can be represented inside the computer with a single bit, which can encode 2 values: black or white. A single dot on the screen is a **picture element**, or **pixel**. A rectangular array of bits is a **bitmap**. A bitmap that is 800 bits wide and 1,000 bit high would contain 800,000 bits or roughly 100,000 bytes or 100 kilobytes. (Actually, a kilobyte is 2^{10} bytes or 1024 bytes.)

Images can also be in shades of gray, or color. The available shades or hues is determined by the number of bits used to represent an individual pixel. This value is known as the **color depth** or bits per pixel. See https://en.wikipedia.org/wiki/Color_depth Using one byte per pixel has a color depth of 8, which could represent up to 256 different colors or shades of gray.

HTML colors used in web pages typically encode colors in RGB format, with either a name, or a triple specifying the saturation of red, green, and blue separately with 8 bits each. Thus, RGB is a 24 bit number, usually represented with a name, like “IndianRed”, or hexadecimal values, like “#CD5C5C”. See https://en.wikipedia.org/wiki/RGB_color_model and https://htmlcolorcodes.com/

Images with 24 bits per pixel can require massive amounts of memory. This is an issue not only for storage in memory or on disk, but also for transmission over the network. The larger the image, the longer it takes to move over the internet. One way to address this issue is to compress the image.

The original graphical web browser was Mosaic, written largely by Marc Andreesen, who was a staff programmer at the University of Illinois at Champaign-Urbana. See https://en.wikipedia.org/wiki/NCSA_Mosaic At the time, 1992, the world wide web was largely text-based. Existing browsers, like lynx, transmitted text only. Andreesen wanted to add graphic images and wanted the browser to run on all three primary platforms, namely, Macs, PC's, and UNIX X-Windows. Unfortunately, each platform had its own graphics format which was not supported by the other two. Why should it? However, there was a popular dial-up information utility, Compuserve, that was cross platform. Compuserve had developed its own format that ran on all three platforms - the graphics interchange format or GIF. See https://en.wikipedia.org/wiki/GIF GIF had an 8 bit color depth, but each image could have a custom color palette chosen from the 24 bit color choices. Andreesen went with GIF for his Mosaic browser. GIF supported animation, transparent layers, and compression. A large GIF image could be compressed with Lempel-Ziv-Welch (LZW) without any loss of image quality. See https://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Welch LZW is a lossless compression algorithm. As such, it can also be used for compressing computer code, where any loss could be fatal.

An alternative to GIF is the JPEG standard, developed in 1992 by the Joint Photographic Experts Group. See https://en.wikipedia.org/wiki/JPEG#:~:text=JPEG Unlike GIF, JPEG is lossy. That is, you cannot take a JPEG image and recreate the original photograph. However, JPEG typically achieves a 10:1 compression that is not detectable by the human eye, which is the typical audience for such images.

The fact is, you can alter an image file drastically without changing how it appears to the human eye. Can you spot the difference in the following two images?

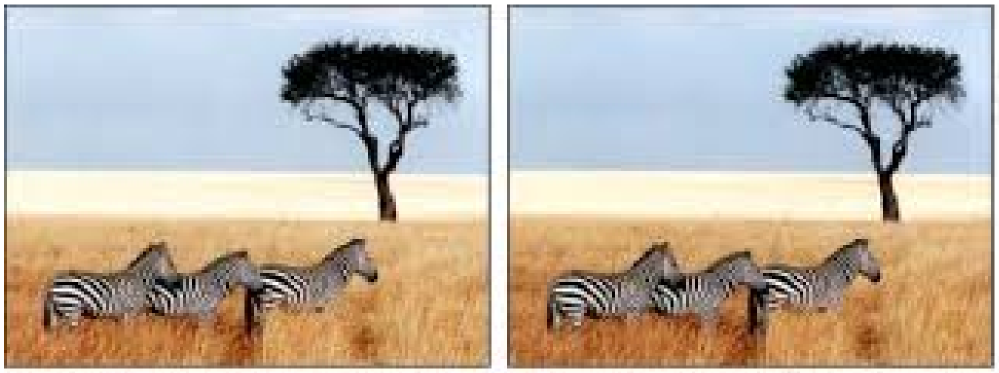


Figure 14: Bitmap (.bmp)files from Tanenbaum demo. The left images is the orig
image contains hidden text (36)

The computer scientist Andrew Tanenbaum is the author of a major text on computer networks. See https://www.cs.csubak.edu/~jyang/Computer-Networks---A-Tanenbaum---5th-edition.pdf He is also an amateur photographer. He took the above zebra picture at the left, and then encoded his entire textbook into the image at the right using *steganography*. See https://en.wikipedia.org/wiki/Steganography

The bottom line: any digital image of any size is simply a string of bits.

Code

When you think of computer programs, you may imagine a source code program, like the following Python procedure which adds 10 to its argument:

```
def add10(n):
      return n+10
```

This code is just a sequence of characters, that is, text. However, when you execute the code, Python must first convert it into byte code, and ultimately, binary machine code. Python lets the programmer peek behind the scenes and see the internal representation.

```
>>> dir(add10)
['__annotations__', '__builtins__', '__call__', '__class__',
'__closure__', '__code__', '__defaults__', '__delattr__', '__dict__',
'__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__get__',
'__getattribute__', '__globals__', '__gt__', '__hash__', '__init__',
'__init_subclass__', '__kwdefaults__', '__le__', '__lt__',
'__module__', '__name__', '__ne__', '__new__', '__qualname__',
'__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__']
>>> dir(add10.__code__)
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__gt__', '__hash__',
'__init__', '__init_subclass__', '__le__', '__lt__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__', 'co_argcount',
'co_cellvars', 'co_code', 'co_consts', 'co_filename',
'co_firstlineno', 'co_flags', 'co_freevars', 'co_kwonlyargcount',
'co_lines', 'co_linetable', 'co_lnotab', 'co_name', 'co_names',
'co_nlocals', 'co_posonlyargcount', 'co_stacksize', 'co_varnames',
'replace']
>>> add10.__code__.co_code
b'|\x00d\x01\x17\x00S\x00'
```

The Python `dir()` procedure lists the properties of Python objects. The procedure `add10` has a __**code**__ property which itself has a `co_code` property, whose value is a byte string, displayed in hexadecimal. Those bytes are Python byte code, which is typically a tuple comprising a one byte operation code and a one byte argument. The Python `dis` module allows us to disassemble the byte code.

```
>>> import dis
>>> dis.dis(add10)
  2           0 LOAD_FAST                0 (n)
              2 LOAD_CONST               1 (10)
              4 BINARY_ADD
              6 RETURN_VALUE
```

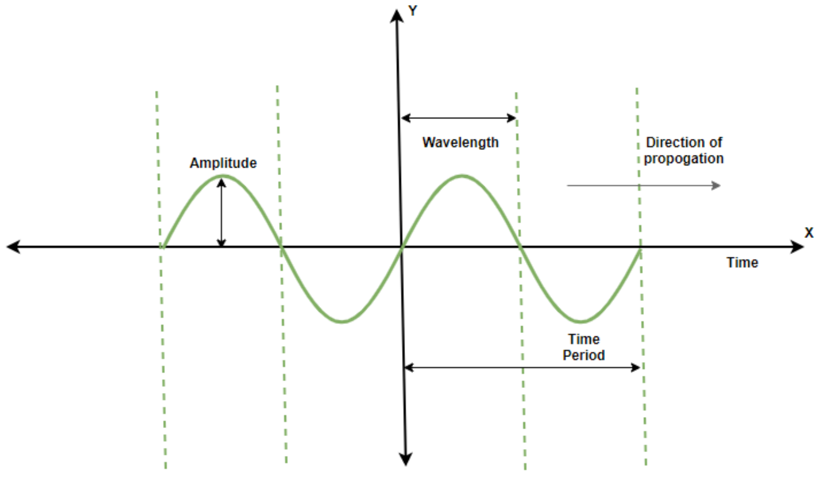
See https://docs.python.org/3/library/dis.html for details about dis and Python byte code.The Python Virtual Machine interprets the byte code and runs the program on the target machine. See https://leanpub.com/insidethepythonvirtualmachine/read which goes into a lot of detail.

The bottom line: any computer program of any size is simply a string of bits.

Sound

We have discussed numbers, text, images, and code. We now examine music, or more generally, sound. How can we represent sound inside a computer?

You may be aware from physics that sound is a wave that travels through a medium, like air, and creates auditory sensations in our ears, which respond to the wave oscillations. A sound wave, or any wave for that matter, has a frequency and amplitude.



See https://www.geeksforgeeks.org/what-are-the-characteristics-of-sound-waves/

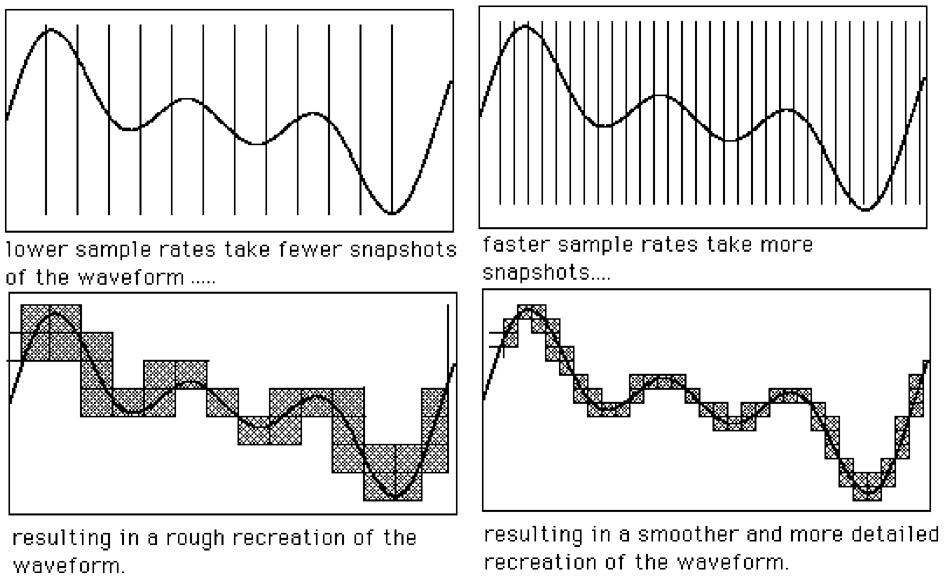
The frequency determines the pitch of the sound. A high frequency has a high pitch and a low frequency has a low pitch. Frequency is measured in cycles per second, which is specified using the scientific unit name hertz, in honor of Heinrich Hertz, a German physicist. Normal human hearing spans the frequency range of 20 to 20,000 hertz. See https://en.wikipedia.org/wiki/Audio_frequency

The amplitude determines how soft or loud the sound is. See https://en.wikipedia.org/wiki/Sound

A sound wave is an analog, not digital phenomenon. It is continuous. Though we did not mention it above, a painting or the subject of a photograph is likewise an analog phenomenon. Taking a digital photograph converts the analog image into thousands of digital pixels.

Traditional telephony was analog as well. The sound from the source was represented as an analog wave that was transmitted along the phone line as an electronic wave and then converted back to sound at the receiver's speaker.

To process digital sound, you first need an analog to digital (A-TO-D) converter, which takes an analog sound wave and converts it into a series of numbers, each of which can be represented digitally. See https://en.wikipedia.org/wiki/Analog-to-digital_converter When you want to play back a digitized sound, you use a digital to analog (D-TO-A) converter which reverses the process. See https://en.wikipedia.org/wiki/Digital-to-analog_converter


lower sample rates take fewer snapshots of the waveform .....
faster sample rates take more snapshots....
resulting in a rough recreation of the waveform.
resulting in a smoother and more detailed recreation of the waveform.

See https://www.animemusicvideos.org/guides/avtechbeta/audio1.html

The analog wave is sampled at regular intervals. The higher the sampling rate, the greater the fidelity of the sound. According to information theory developed by Claude Shannon at Bell Laboratories, the sampling rate must be twice that of the highest frequency you wish to represent. See https://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem Note that Shannon made significant contributions to many fields. He was among the original AI and cryptography researchers. See https://en.wikipedia.org/wiki/Claude_Shannon He has been compared to both Einstein and Newton. The AI chatBot Claude from Anthropic is named to honor Shannon. See https://claude.ai/ Note that Yale's Clarity AI platform now includes Claude as an option, along with ChatGPT.

As stated above, the upper end of human hearing is 20,000 hz. Given the sampling theorem, we conclude that the sampling rate for digital sound should be at least 40,000 hz. In fact, audio compact disc encoding samples at 44,100 hz. See https://en.wikipedia.org/wiki/Compact_Disc_Digital_Audio   This means that one minute of CD quality sound has 60*44,100 or 2,646,000 samples, or twice that for stereo.

That's a boatload of data. The audio standard for the iPod and most computer music is MP3, which uses lossy compression like JPEG, to achieve better storage and transmission limits. See https://en.wikipedia.org/wiki/MP3

In the early days of the internet, it was common to share digital sound files. In 1993 and 1994, there was a program called "Geek of the Week" which was similar to a podcast. Each week a different computer scientist would be interviewed and a copy of the digital audio file was available for download. See https://town.hall.org/radio/Geek/ However, this was pre-streaming. You had to download the entire file before you could play it back.   With streaming, you can begin playback before the download is complete. This is a huge win, not only for sound, but also for video. See https://en.wikipedia.org/wiki/Streaming_media

The bottom line: any digital sound of any size is simply a string of bits.

Video

We can digitize images and sound. Put that together and we have digitized video. Well, that's pretty much correct.

There is more to it in order to make it feasible to download or stream vast amounts of video data. As with images and sound, compression is important. Also, even though most video comprise 24 or 30 frames a second, in practice there is very little that changes from one video image to another. Thus, video streaming needs to capture only the changes, not entire images.

The video encoding standard is the Motion Picture Expert Group or MPEG. See https://en.wikipedia.org/wiki/Digital_video

The bottom line: any digital video of any size is simply a string of bits.

Music

We have argued that all data processed by a computer can be viewed simply as a string of bits. Theoretically, all that any computer program does or can do is to convert one string of bits into another string of bits.

So what? It sounds like describing all food in terms of calories and nutrients. While that may be correct, we generally are concerned with other properties of food, such as flavor and taste.

We have shown that sound, and thereby music, can be represented by a string of digital samples, usually 44,100 samples per second. We now examine music through another process of abstraction.

Most musicians do not discuss frequencies or hertz, though the orchestra may tune to the pitch of A 440, which is the note A with a frequency of 440 hz.

Musicians discuss notes and scales and arpeggios and chords and harmonies and melodic lines. An octave in music is actually an interval in which the higher note is twice the frequency of the lower note. The other common intervals have similar mathematical properties. These relations are clearly on view in a pipe organ, where the length of a pipe is inversely proportional to its frequency. The longer pipes have lower pitches.

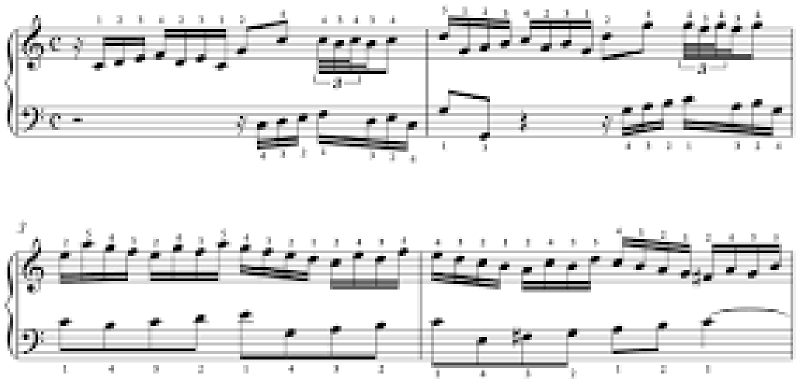
See https://www.letourneauorgans.com/publication/general-information-about-pipe-organs

As an aside, Donald Knuth, the father of complexity theory as well as the text formatting systems TeX and Metafont, is himself an accomplished organist. He used the proceeds from his monumental set of books: *The Art of Computer Programming* to purchase a pipe organ for his home. See https://en.wikipedia.org/wiki/The_Art_of_Computer_Programming and https://www-cs-faculty.stanford.edu/~knuth/organ.html

- Music is like a computer program.
- Musical forms are like programming design patterns.

Full disclosure, the author spent many teenage hours restoring pipe organs as a member of the American Theatre Organ Society in Atlanta, home to the Fox Theatre's 3,622 Möller pipe organ, Mighty Mo. See https://en.wikipedia.org/wiki/American_Theatre_Organ_Society and https://www.atlantamagazine.com/news-culture-articles/what-makes-the-mighty-mo-the-fox-theatres-organ-so-special/

A musical composition is not unlike a computer program. Sheet music, like computer code, specifies a sequence of events. Below are the opening 4 measures of Bach's 2 part Invention number 1, in C major.



It is a sequence of notes that has a starting point and a conclusion, just like a computer program is a series of instructions that has a starting point and (hopefully) halts. Most computer programs have a single thread of execution like a melody. However, it is becoming more common to have parallel computer programs that have multiple threads of execution that are processed in parallel, like the two parts in the above invention.

Beyond the time-related dimensions of music, like note duration and tempo, music has tonal and harmonic characteristics, such as major and minor scales, chords, and progressions of chords. Each of these elements is an abstraction that can be decomposed into its elements, much like computer programs can be decomposed into instructions, byte codes, and ultimately, strings of bits.

At a higher level of abstraction, composers use various musical forms to organize their ideas. Common forms include fugues, theme and variation, rondos, sonata allegro, and dances, such as a waltz or tango.

Computer programs similarly have forms that organize common and useful structures. See https://en.wikipedia.org/wiki/List_of_abstractions_(computer_science) There is actually a hierarchy of abstractions. At a low level, there are entities such as variables, functions, algorithms, interfaces. Next you have data structures, such as lists, stacks, queues, trees,hash tables, heaps, and arrays. Functional and concurrent programming have their own abstractions, as does networking. Programming design patterns, like musical forms, provide structures for organizing code. Examples include the real-eval-print loop used in interpreted languages like Python or Racket, or for that matter, the Google search bar interface.