CSE 101

Computer Science Principle

Lecture 02: Computer History & Computational Thinking

March. 2023
Prof. Francois Rameau
francois.rameau@sunykorea.ac.kr

SUNY Korea
The State University of New York

To reply questions from the previous class

- I -

Are the marking relative or absolute?

It will be absolute

- II -

Is this class similar to Harvard CS50

Indeed, the content will be close to what is taught in CS50. Some explanations in this lecture might even be inspired by it!

Watching CS50 might be a very good complement to this lecture

- III -

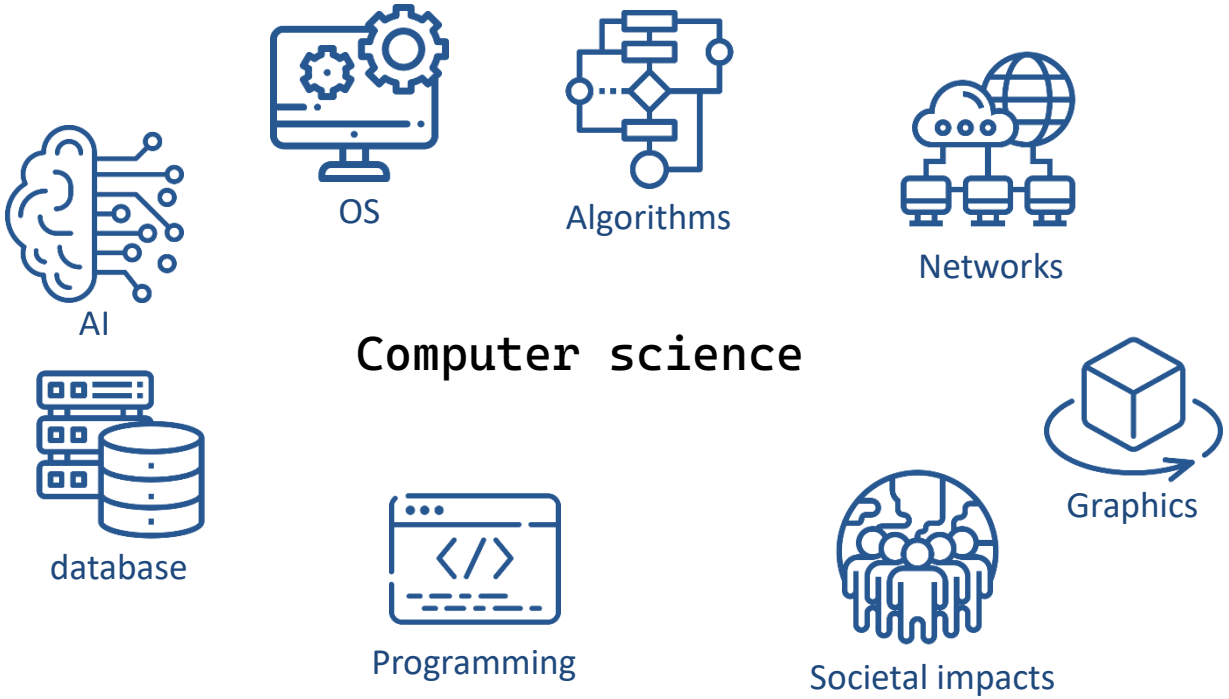
Revised grading scale

A = 93-100; A- = 90-92; B+ = 87-89; B = 83-86; B- = 80-82; C+ = 77-79; C = 73-76; C- = 70-72; D+ = 67-69; D = 63-66; D- = 60-62; F = 0-59

What is Computer Science?

Computer science is all about using computers and computing technology to solve challenging, real-world problems in fields like science, medicine, business and society

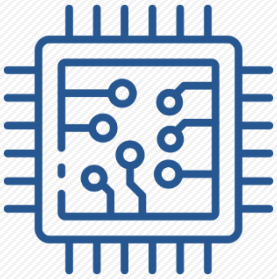
Computer Science ≠ Computer programming



Computing systems

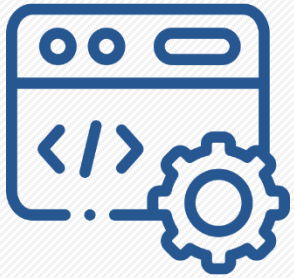
A computing system is composed of two major parts

Hardware



- Screen, keyboard, mouse
- Central Processing Unit (CPU)
- Hard drives & other storage unit

Software



- Application software, like office productivity programs, video games, web browser
- System software, like OS, database systems

Computing systems

Can hardware exist without software?

Yes (in certain case), but is it useful?

- For general purpose CPU (intel x86, core i7, etc.), not really
- Specialized hardware (FPGA, ASICs with functionality built into harware) – Yes. However, their functionality was developed with software tools

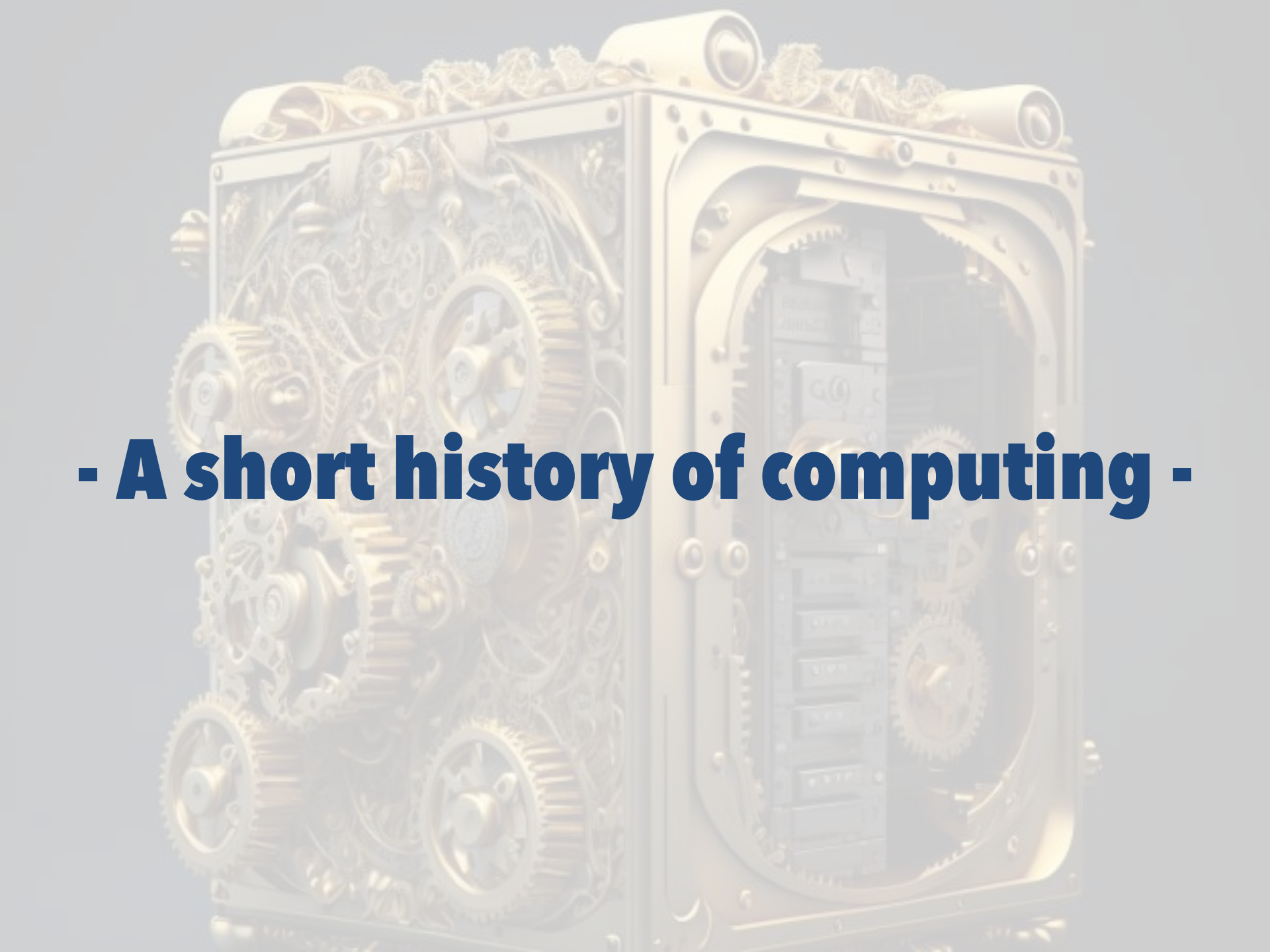
Can software exist without hardware?

Yes (in certain case), but is it useful?

- In a literal sense, no – hardware is needed to execute software
- But, the problem-solving techniques used by programmers to create software do exist separately from the hardware and software

One more part to a computer system: data

The software needs some kind of data to process: numbers, text, images, sound, video



- A short history of computing -

The first computing device?

What do you see when you think about computer?

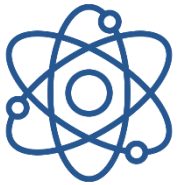
The first computing device?

Calculating has always been essential even in ancient civilization

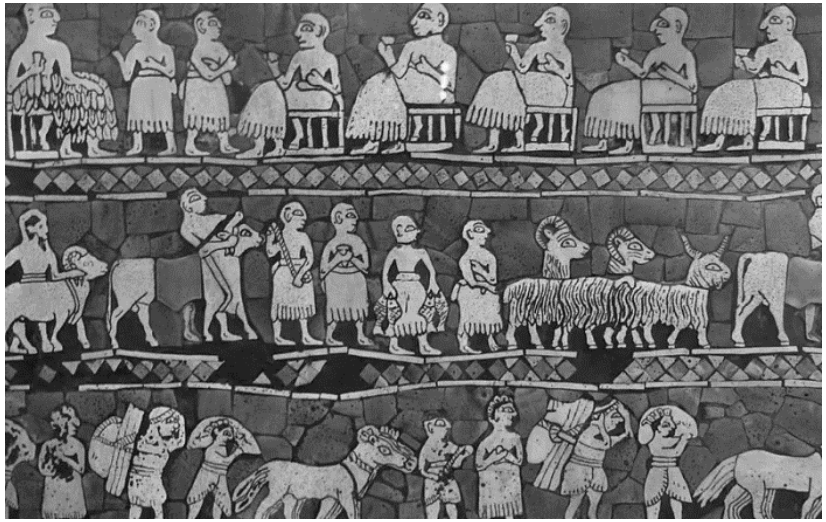
Agriculture

Trade

Architecture

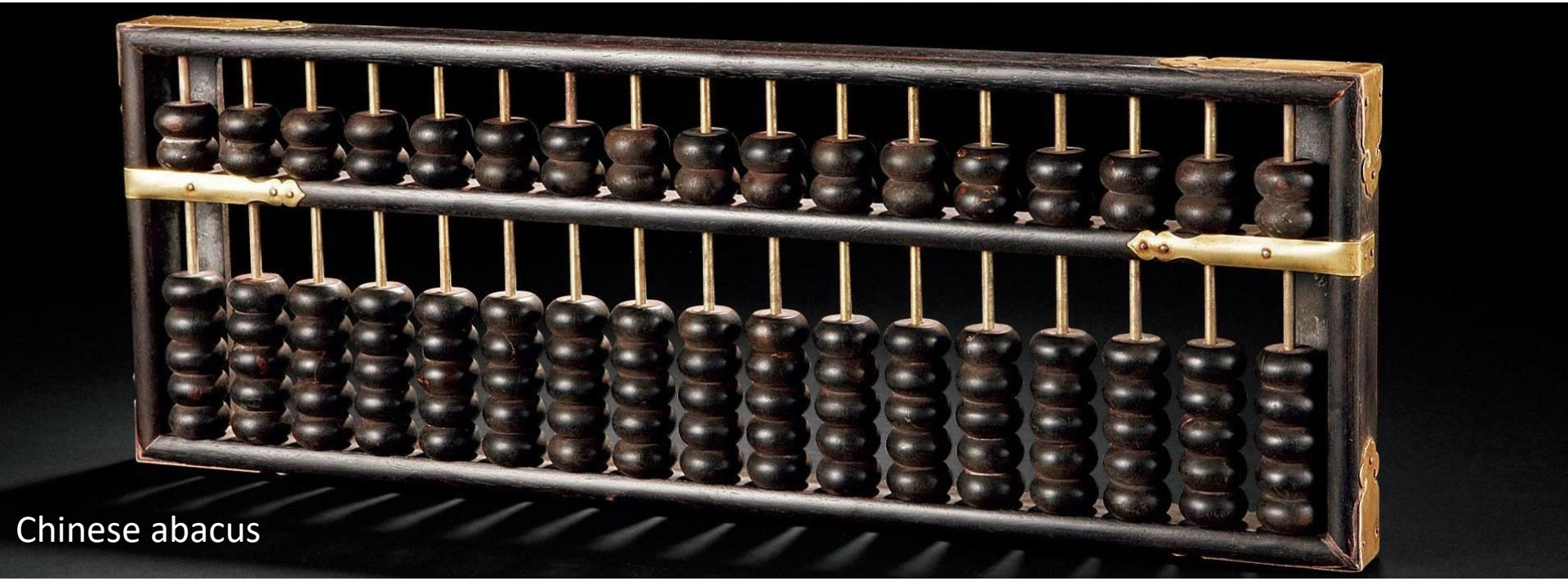


Science



The first computing device?

Computing devices have been developed thousands years ago


Chinese abacus

The **abacus** is an ancient manual counting (Sumerian 2700BC) tool for performing quick arithmetic operations (summation, subtraction, multiplication and division)

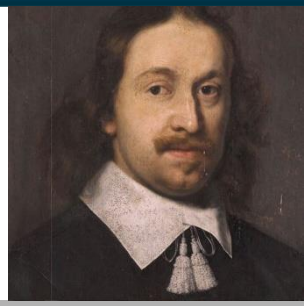
Fun note: the Sumerians were working with a base 60

The first computing device?

Modern computers share 4 main central concepts with the abacus

	1. Storage	2. Data Representation	3. Calculation	4. User Interface
Abacus	An abacus stores numbers, the most fundamental data type in modern computing	The abacus represents numbers using beads on spindles	By moving beads on abacus spindles, user can perform addition, subtraction, multiplication, and division	The beads and spindles in the abacus
Computers	In a modern computer, all data – text, images, audio, video – is represented using binary numbers (1s and 0s)	Computers employ a variety of techniques for representing data on storage media: Magnetic (HDD), electrical (SSD), optical	Modern computers contain powerful central processing units that perform calculations at astonishing speeds	Modern computers provide a wide variety of input and output devices for the user

The Pascaline



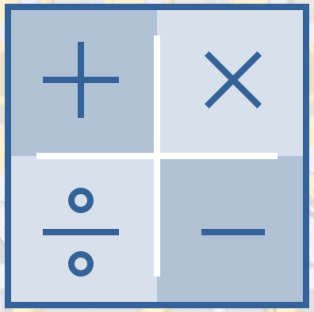
During the 17th century, **Blaise Pascal** built the Pascaline to help his dad (tax receiver)

- Mechanical calculator that could only do addition and subtraction
  - Input is given using dials
  - Output is read on small windows above each dial
  - Can eventually do division and multiplication by repetition
- Not programmable!
- Mathematicians were very afraid! "Are we going to lose our jobs?"

The Leibniz calculator

Inspired by Pascal, **Leibniz** made his own calculating machine

- Able to perform all four arithmetic operations

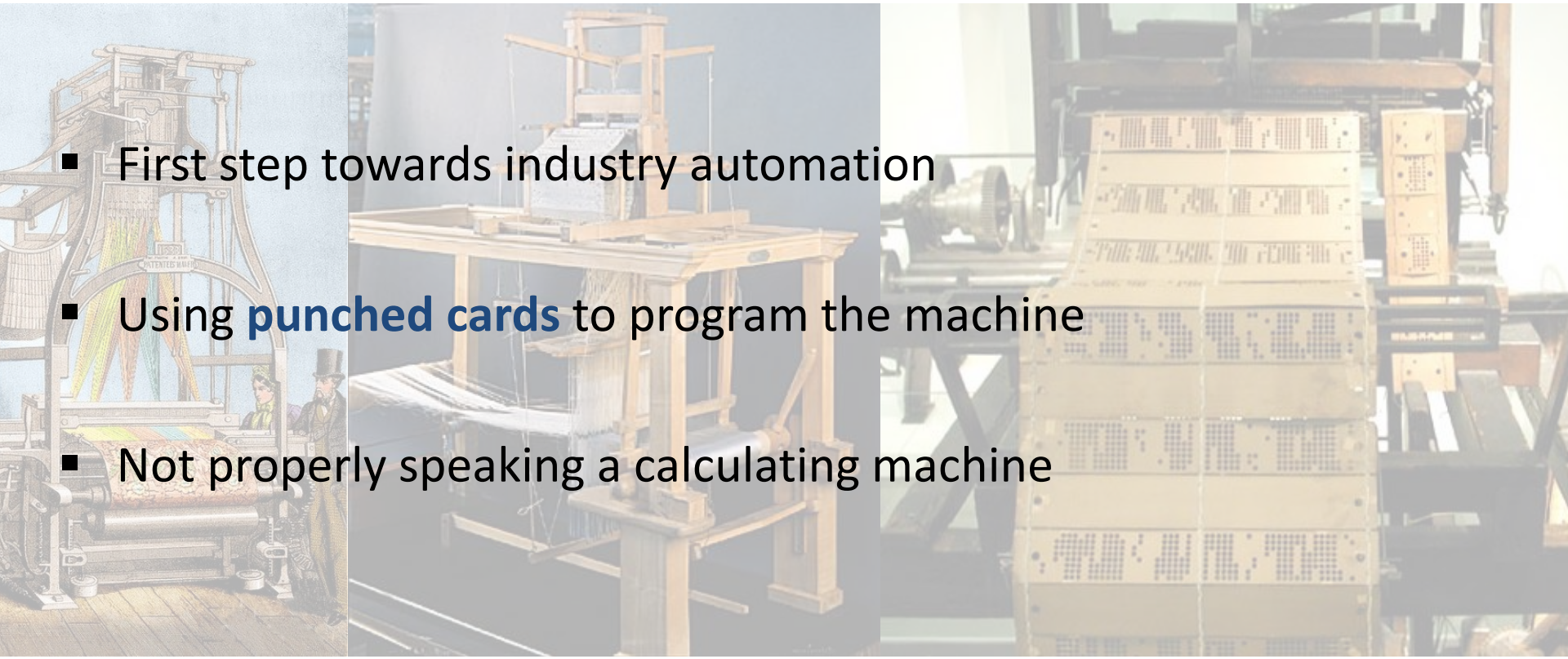


- **Still not programmable**

"It is unworthy of excellent men to lose hours like sla ves in the labour of calculation which could safely be relegated to anyone else if machines were used."

Programmable devices

The first programmable machine was a loom invented by **Jacquard** in 1804

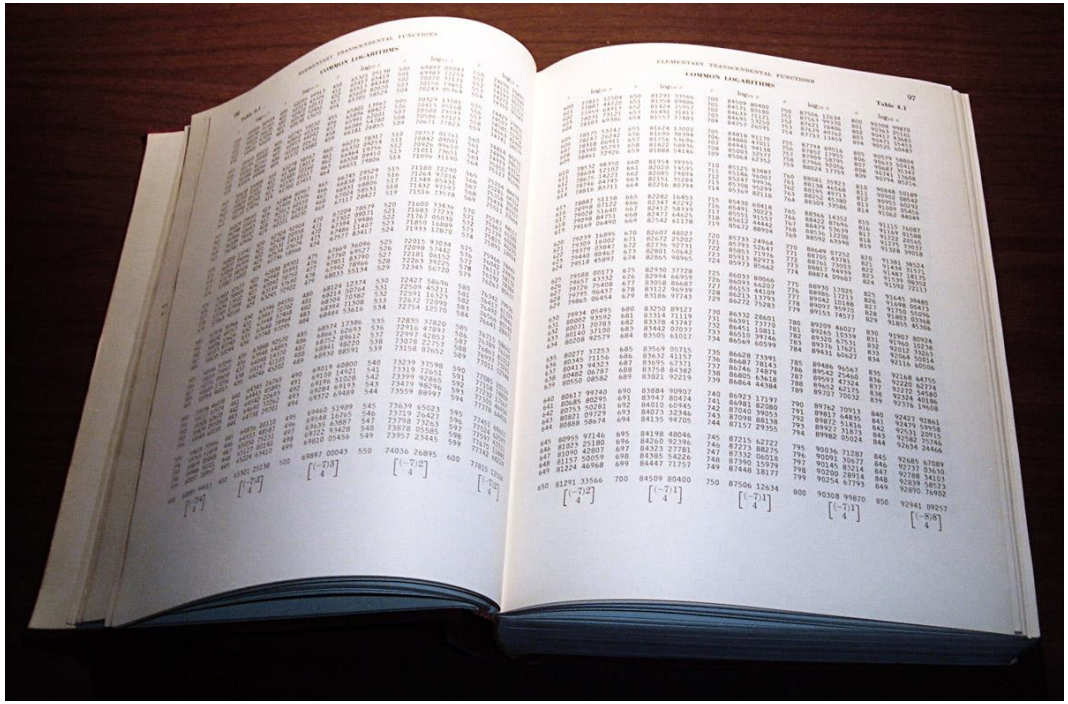


- First step towards industry automation
- Using **punched cards** to program the machine
- Not properly speaking a calculating machine

Pre-computed table

These calculating machine **were expensive, slow and limited**

- For an extended period of time scientists and engineer relied on pre-computed calculation table

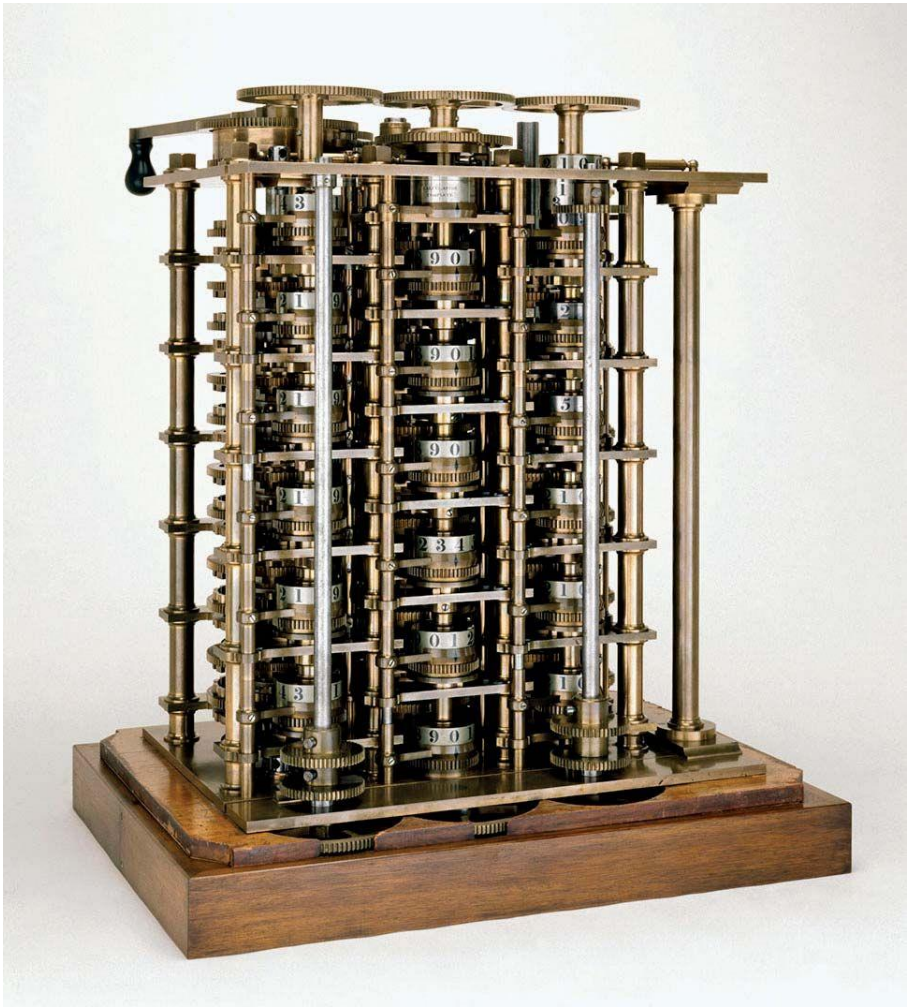


Programmable devices

The ancestor of the computer has been proposed by Babbage (able to resolve polynomial equations)

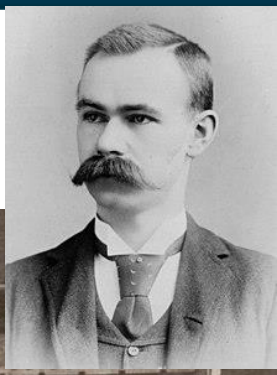
He designed the Analytical Engine – a mechanical, programmable computer

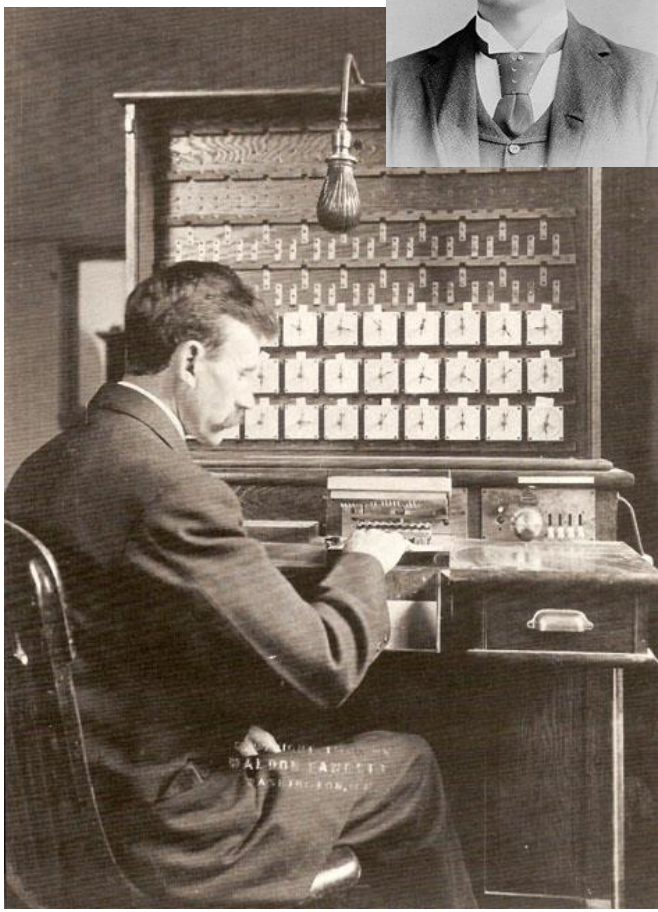
- It was never actually built in Babbage's time due to a lack of manufacturing capabilities
- He noticed that many computations consisted of operations which were regularly repeated
- Design called for punched cards to be fed into the machine to program it to perform mathematical calculations
- Output would go to a printer or more punched cards



From mechanical to electro-mechanical

Following the work of Babbage, **Herman Hollerith** developed the Census tabulator (1890)





- First electro-mechanical computer
- Read data from punched cards
- Used for American census
- His company became:

IBM

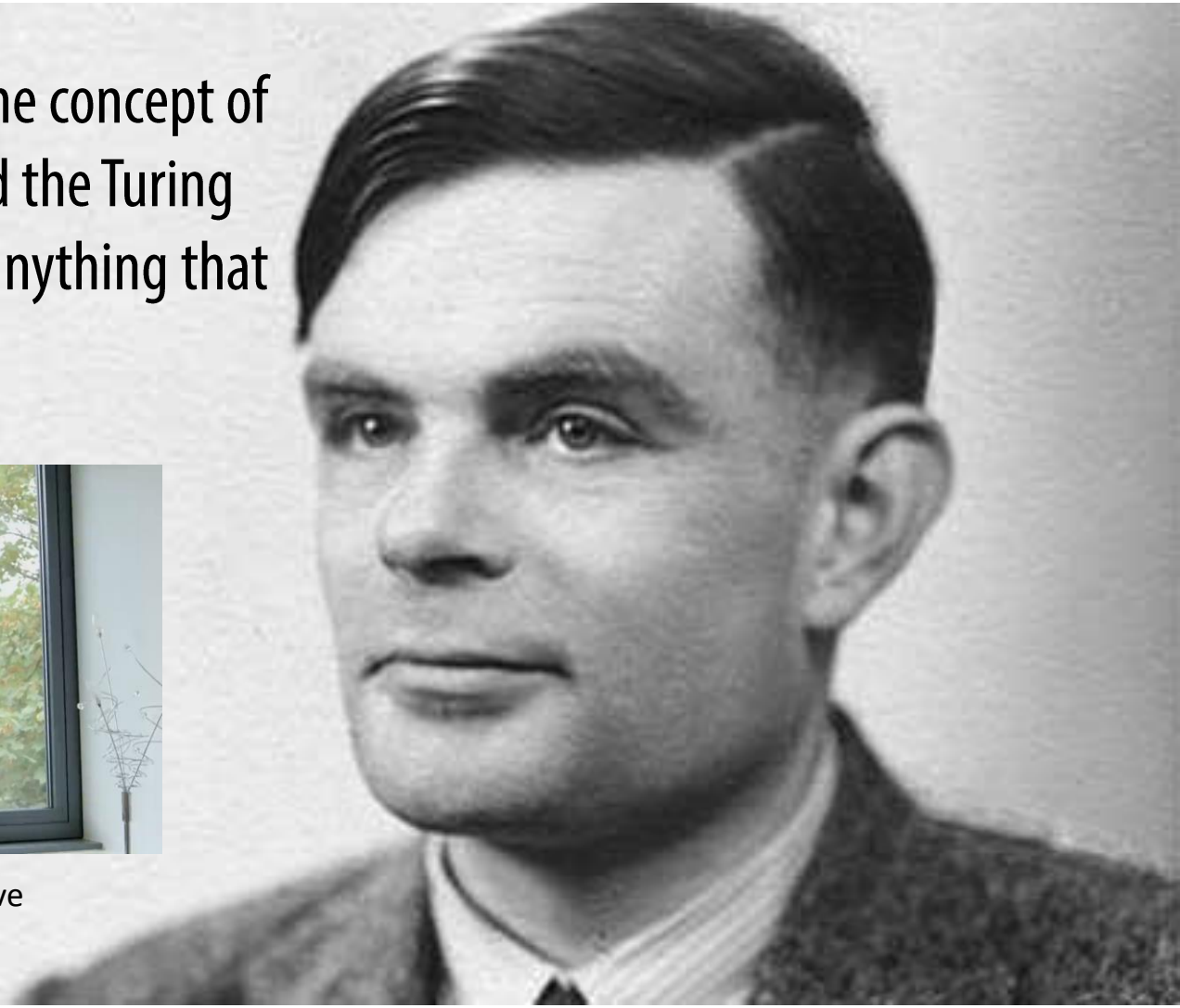
Human computers

Originally a computer was a person performing computation

The Turing machine

In 1936, **Turing** proposed the concept of a universal machine called the Turing machine, able to compute anything that is computable

Let's watch it if we have enough time (5min)



Programmable computers

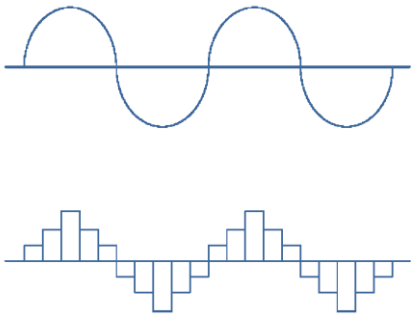
Now, move forward to the 20th and 21st centuries

A modern computer is defined by three basic requirements

1. Must be electronic and not exclusively mechanical.

2. Must be digital, not analog

   Digital devices use discrete values (digits), not a continuous range of values to represent data. (i.e. digital vs mercury-based thermometer)



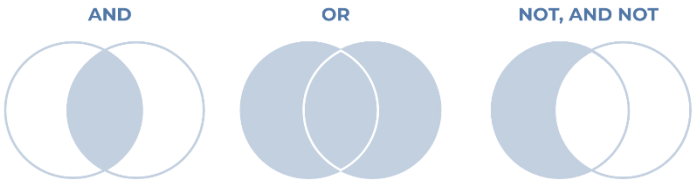
3. Must employ the **stored-program concept**

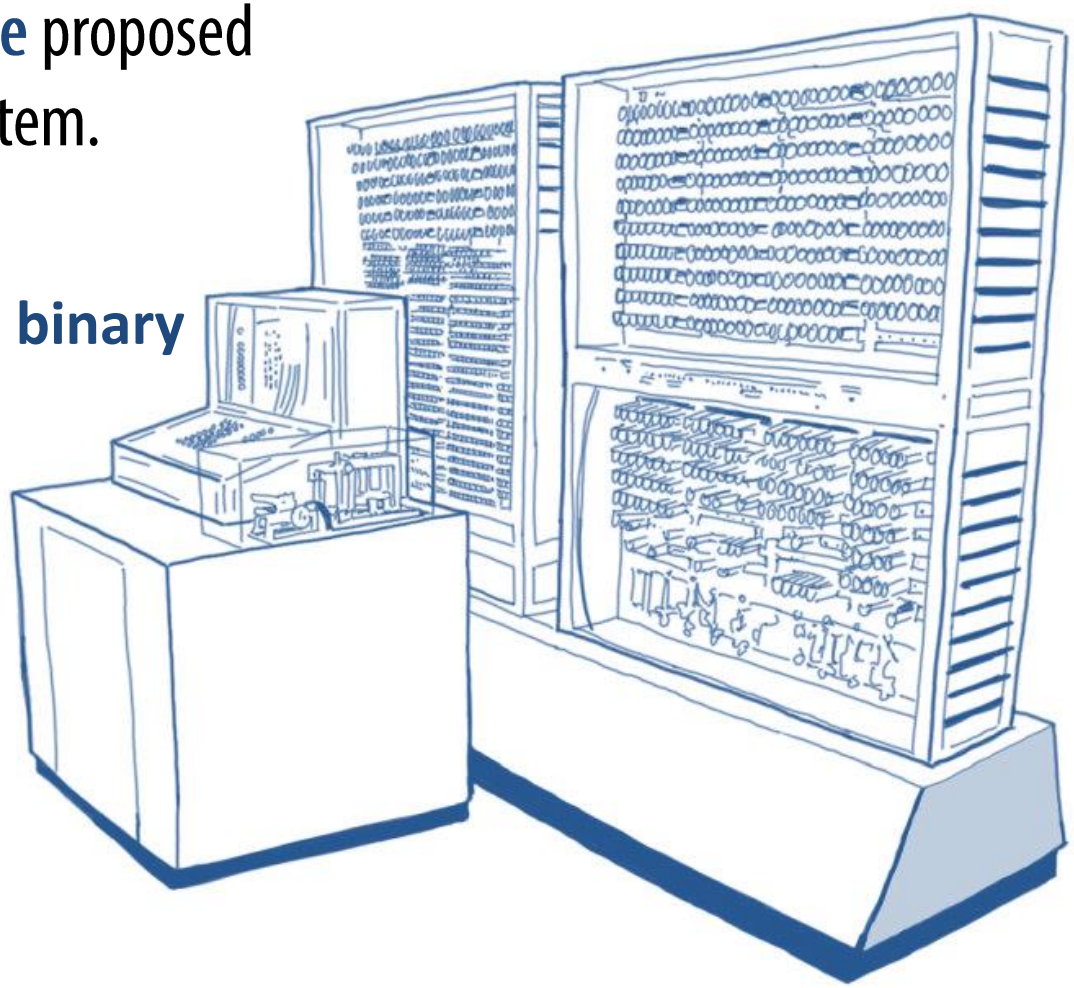
   The device can be reprogrammed by changing the instructions stored in the memory of the computer

Towards Boolean logics computers

Unlike its predecessor **Konrad Zuse** proposed to work with the binary system.

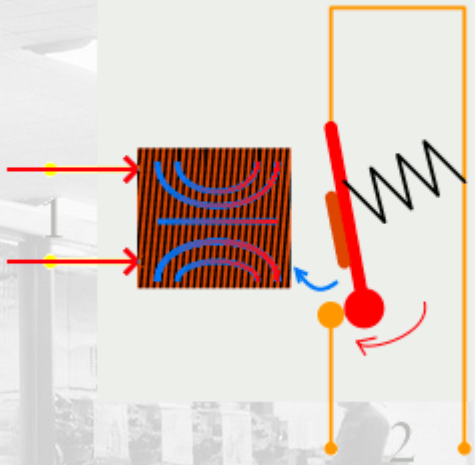
- The machine was using the **binary** system and **Boolean logic**
- First commercial computer





Mark I: computing at scale

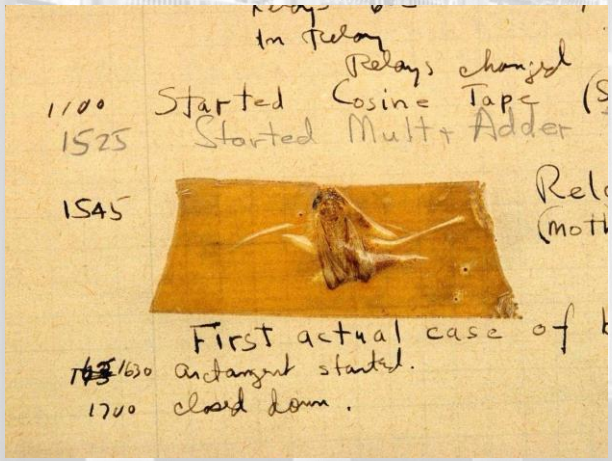
- **Weight**: approximately 5 tons (10,000 pounds)
- **Size**: about 51 feet long and 8 feet high, occupying a room roughly the size of a tennis court
- **Processing speed**: about 3 additions or subtractions per second
- Word length: 23 decimal digits (plus a sign), or 72 binary bits
- **Memory capacity**: 72 words (each 23 digits long), stored on rotating electromagnetic drums
- **Power consumption**: about 5 kilowatts

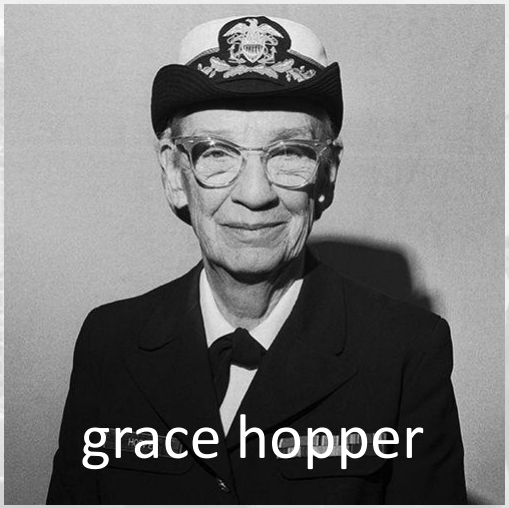


Mostly relying on relay which are slow to flip!
Also prone to wear … among 3500 relays, they were commonly out of service

Mark I: computing at scale

First reported computer bug in history





A substitute to relay?

The vacuum tube

- Invented in the early 20th century
- The vacuum tube can be used to replace the relays in computers
- Faster computation
- More robust (no mechanical elements)

ENIAC

The first electronic, general-purpose, programmable computer: ENIAC (1946)

Electronic Numerical Integrator And Computer

- Among the first computers to employ the stored-program concept
- Much faster than any of its predecessor
- Worked for 10 years and supposedly did more arithmetic operation than the entire human race at this point

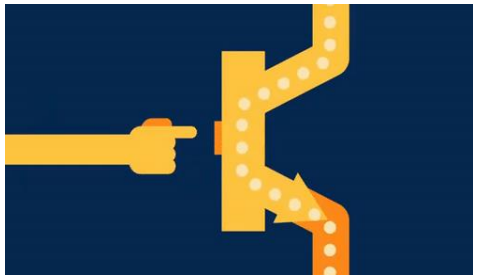
Modern computers have four major components:

- Input device(s)
- Output device(s)
- Memory – for data storage, both temporary & permanent
- Processor – for doing computations

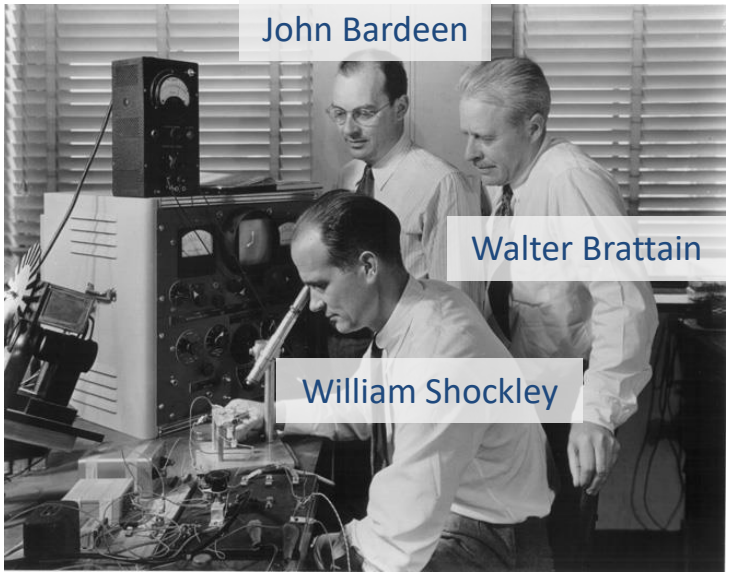
The transistor

Vacuum tubes have their limitations (bulky, expensive, not robust)

In 1947 **transistors** came to the rescue!



The exact functioning principle of transistor is complex, but you can assume it acts as a relay.



Transistor was a groundbreaking discovery (one of the most important for the humankind):

- A Solid State Component (SSD)
- The first prototype could be switched ON and OFF 10,000 times per second (10,000Hz)
- Can be made very small
- Much cheaper, made of sillicon

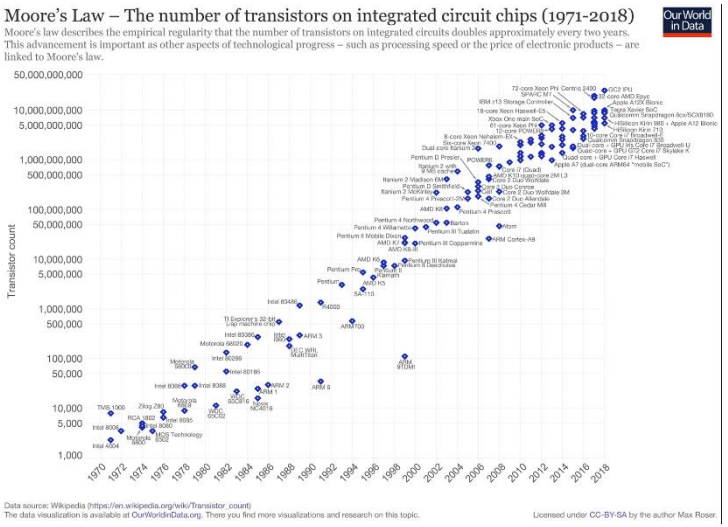
The age of transistor

Moore's law states that the number of transistors on a microchip doubles about every two years, resulting in exponential growth of computing power.

Transistor size:

- 1950s-1960s: A few millimeters in size
- 1970s-1980s: A few micrometers (microns) in size
- 1990s-2000s: A few hundred nanometers (nm) in size
- 2010s-2020s: A few tens of nanometers in size (around 10-20nm but can be as small as 3nm)

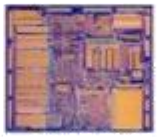
They are not only small, they are very fast and can switch millions of times per second and last decades




1950s
Silicon Transistor
1 Transistor


1960s
TTL Quad Gate
16 Transistors


1970s
8-bit Microprocessor
4500 Transistors


1980s
32-bit Microprocessor
275,000 Transistors


1990s
32-bit Microprocessor
3,100,000 Transistors


2000s
64-bit Microprocessor
592,000,000 Transistors

Note: Due to physical limitations, Moore's law will certainly not be verified in the near future.
Multicore and parallelism compensate for that.

- Binary & Boolean arithmetic –

There are 10 types of people in this world, those who understand binary and those who don’t

Decimal system

What do you read here?

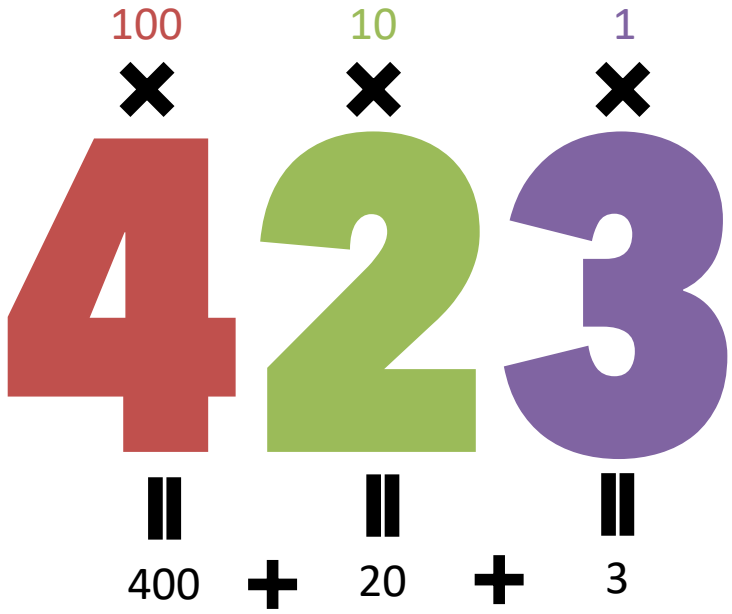
423

Four hundred twenty-three ?

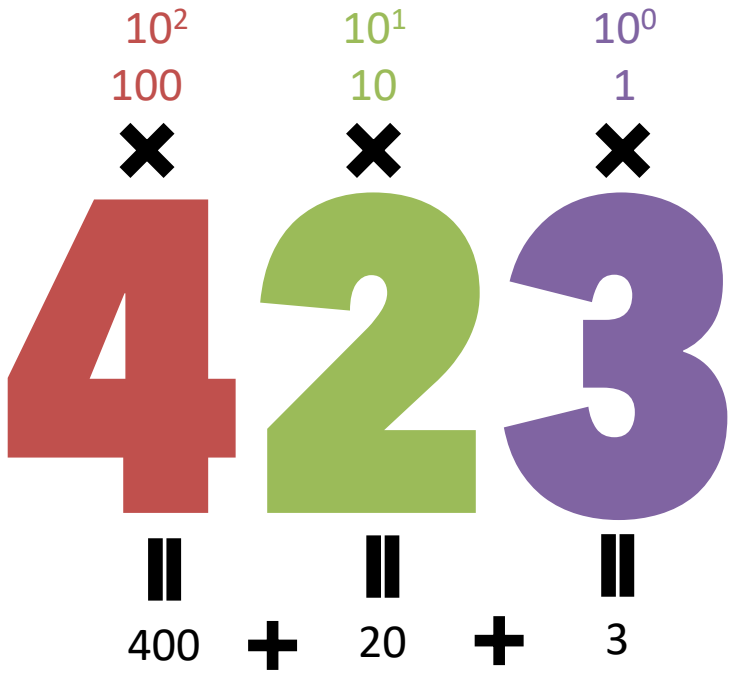
The decimal system is a positional notation numeral system that uses the base of 10. It represents numbers using combinations of digits 0-9, with each digit's value determined by its position in the number.

Symbol										Used with scripts	Numerals
0	1	2	3	4	5	6	7	8	9	many	Arabic numerals
০	১	২	৩	৪	৫	৬	৭	৮	৯	Bengali–Assamese	Bengali numerals
੦	੧	੨	੩	੪	੫	੬	੭	੮	੯	Gurmukhi	Gurmukhi numerals
૦	૧	૨	૩	૪	૫	૬	૭	૮	૯	Gujarati	Gujarati numerals
୦	୧	୨	୩	୪	୫	୬	୭	୮	୯	Odia	Odia numerals
᱐	᱑	᱒	᱓	᱔	᱕	᱖	᱗	᱘	᱙	Santali	Santali numerals
𑇐	𑇑	𑇒	𑇓	𑇔	𑇕	𑇖	𑇗	𑇘	𑇙	Sharada	Sharada numerals
௦	௧	௨	௩	௪	௫	௬	௭	௮	௯	Tamil	Tamil numerals
౦	౧	౨	౩	౪	౫	౬	౭	౮	౯	Telugu	Telugu script § Numerals
೦	೧	೨	೩	೪	೫	೬	೭	೮	೯	Kannada	Kannada script § Numerals
൦	൧	൨	൩	൪	൫	൬	൭	൮	൯	Malayalam	Malayalam numerals
෦	෧	෨	෩	෪	෫	෬	෭	෮	෯	Sinhala	Sinhala numerals
၀	၁	၂	၃	၄	၅	၆	၇	၈	၉	Burmese	Burmese numerals
๐	๑	๒	๓	๔	๕	๖	๗	๘	๙	Tibetan	Tibetan numerals
᠐	᠑	᠒	᠓	᠔	᠕	᠖	᠗	᠘	᠙	Mongolian	Mongolian numerals
០	១	២	៣	៤	៥	៦	៧	៨	៩	Khmer	Khmer numerals
๐	๑	๒	๓	๔	๕	๖	๗	๘	๙	Thai	Thai numerals
໐	໑	໒	໓	໔	໕	໖	໗	໘	໙	Lao	Lao script § Numerals
᮰	᮱	᮲	᮳	᮴	᮵	᮶	᮷	᮸	᮹	Sundanese	Sundanese numerals
꧐	꧑	꧒	꧓	꧔	꧕	꧖	꧗	꧘	꧙	Javanese	Javanese numerals
᭐	᭑	᭒	᭓	᭔	᭕	᭖	᭗	᭘	᭙	Balinese	Balinese numerals
٠	١	٢	٣	٤	٥	٦	٧	٨	٩	Arabic	Eastern Arabic numerals
۰	۱	۲	۳	۴	۵	۶	۷	۸	۹	Persian / Dari / Pashto	
٠	١	٢	٣	٤	٥	٦	٧	٨	٩	Urdu / Shahmukhi	
፩	፪	፫	፬	፭	፮	፯	፰	፱	፲	Ethio-Semitic	Ge'ez numerals
〇	一	二	三	四	五	六	七	八	九	East Asia	Chinese numerals

Decimal system



Decimal system



But a computer made of transistors can only understand successions of 0 and 1;
how to represent numbers for calculation?

Binary system

Binary system

Why binary? Because two states

0 **1**

False
(no current flowing in the circuit)

True
(current flowing in the circuit)

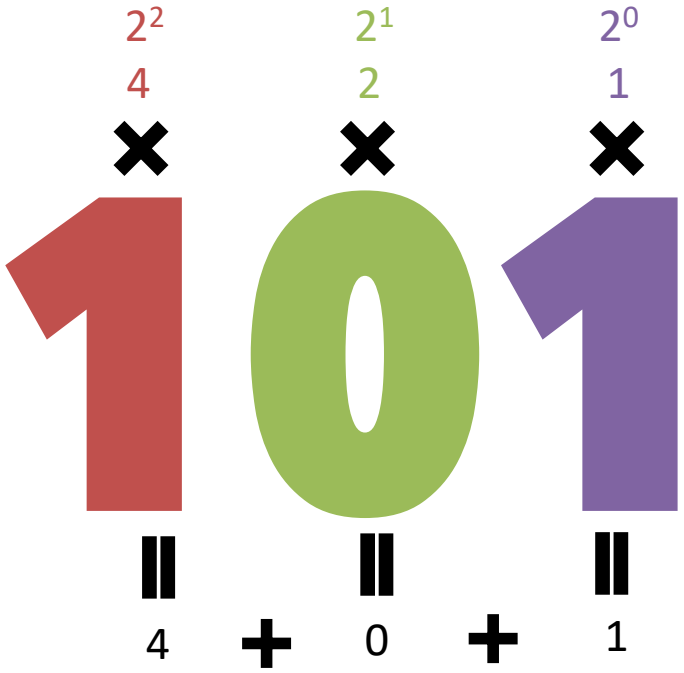
Why only two states?

- Cheaper and easier to build
- More robust to noise
- A very convenient branch of algebra dedicated to it: **Boolean algebra**

With this simple representation we can store image, video music. How is it possible?

Binary system

Base two system



5

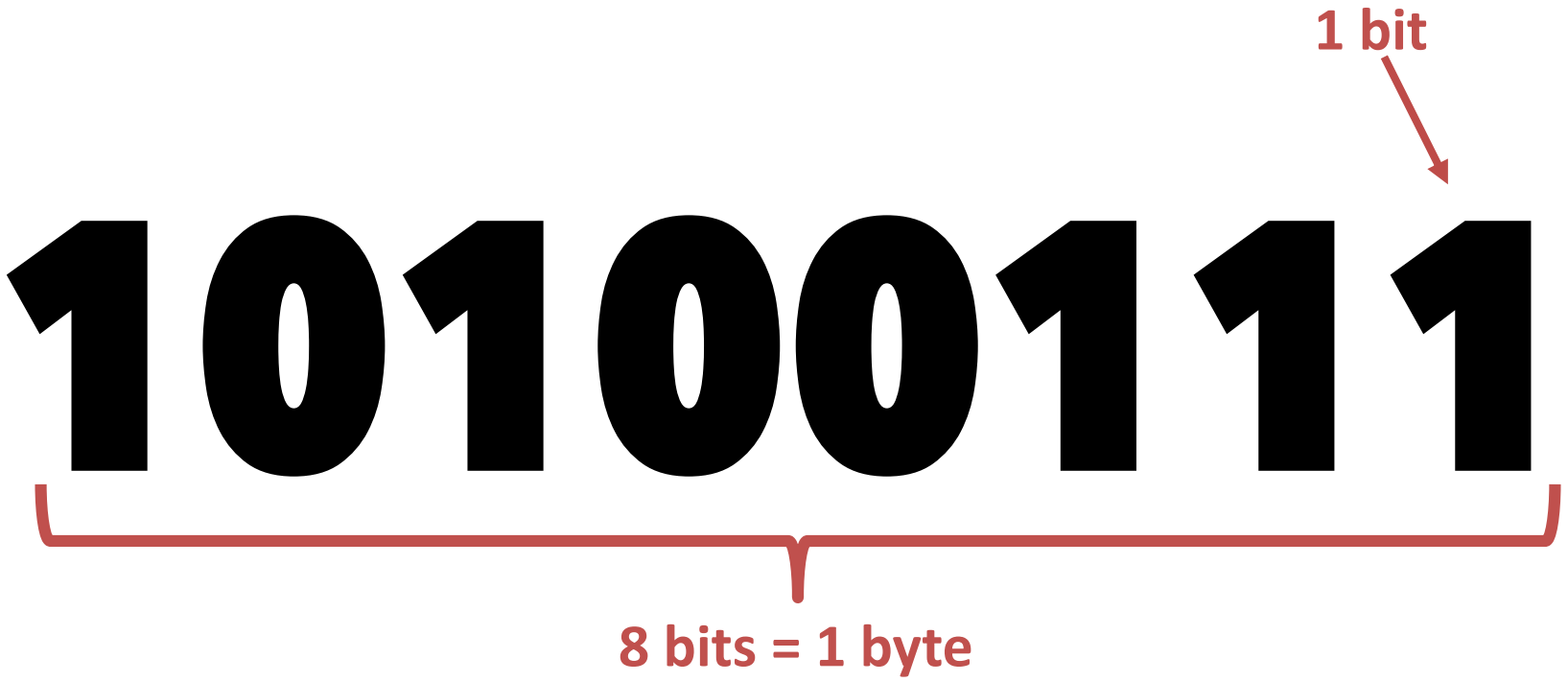
Binary system

Let's challenge yourself!

Binary	**0101**	**1111**	**0011**	**1001**
Decimal				

Binary	**10101**	**00001**	**11100**	**01100**
Decimal				

Binary: a bit of terminology



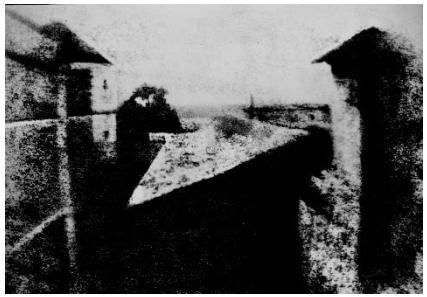
- A file of 1kBytes is composed of $1000 \times 8 = 8000$ bits
- The maximum value that can be coded on a byte is 255

Binary: encoding floats and negative

C Basic Data Types	32-bit CPU		64-bit CPU	
	Size (bytes)	Range	Size (bytes)	Range
char	1	-128 to 127	1	-128 to 127
short	2	-32,768 to 32,767	2	-32,768 to 32,767
int	4	-2,147,483,648 to 2,147,483,647	4	-2,147,483,648 to 2,147,483,647
long	4	-2,147,483,648 to 2,147,483,647	8	- 9,223,372,036,854,775,808- 9,223,372,036,854,775,807
long long	8	9,223,372,036,854,775,808- 9,223,372,036,854,775,807	8	9,223,372,036,854,775,808- 9,223,372,036,854,775,807
float	4	3.4E +/- 38	4	3.4E +/- 38
double	8	1.7E +/- 308	8	1.7E +/- 308

Representing complex data?

We saw that we can represent numbers but a computer can store much more complex information. How does it work?

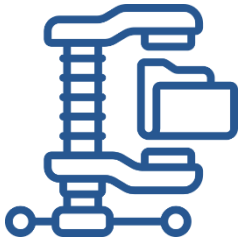


Images

Sound

Video

Documents



Archive files

And much more

Representing complex data?

Letters

To represent letters, we can assign one value for each letter in the alphabet for instance.
With 26 letters in the alphabet, how many bits would you need to code them all?

$$2^5 = 32$$

Our representation:

A = 00000, B = 00001, C = 00010, D = 00011 … Z = 11001

Representing complex data?

Letters

In practice, we often have much more than 26 letters to code. What about accents, special characters, etc. . . A famous example is the **ASCII (American Standard Code for Information Interchange)** which requires 7 bits

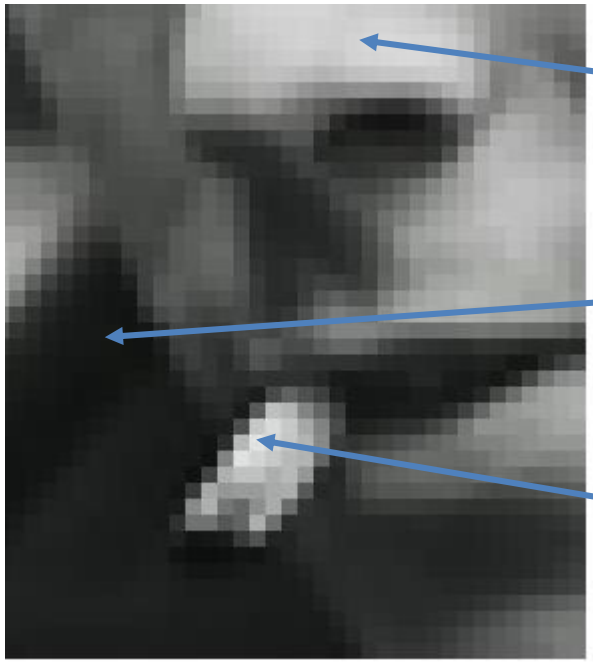
Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	"	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	'	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	|
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

Bonus video about UTF-8

Representing complex data?

Images

- An ensemble of pixels (picture elements)!
- Each cell of the sensor measures how much light it receives



- Pixel values: [0 255] (1 byte)
- The computer sees only numbers!

Representing complex data?

Images

For color you have 3 bytes per pixel to represent three primary colors

Red Green Blue

For instance, a purple pixel can take the values

128 100 162

01000000 1100100 10100010

Representing complex data?

Images

Images can be coded with less bits




Boolean algebra

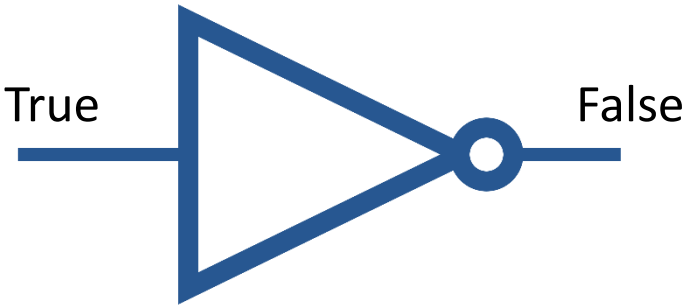
Using binary numbers, a computer is able to perform a vast array of operations, all of which are based on **Boolean algebra**

Three fundamental operations that can easily be implemented with transistors

NOT **AND** **OR**

Boolean algebra: Not

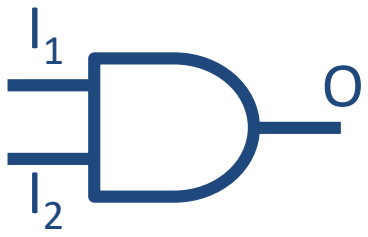
NOT inverses the input value



Input	Output
True	False
False	True

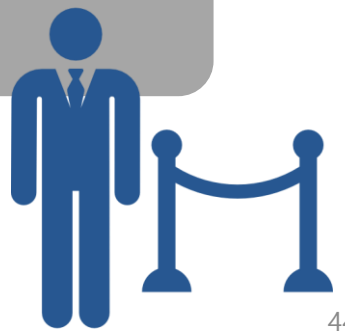
Boolean algebra: And

AND takes two inputs and return a 1 only if the two inputs are true



I_1	I_2	Output
False	False	False
False	True	False
True	False	False
True	True	True

An AND gate is like a bouncer at a nightclub - it won't let you in unless you have your ID **and** dancing shoes.



Boolean algebra: OR

An OR gate is a logical gate that produces an output of 1 (true) if at least one of its inputs is 1 (true), and an output of 0 (false) if both inputs are 0 (false).

I_1	I_2	Output
False	False	False
False	True	True
True	False	True
True	True	True

Imagine you have a security system for your home with two sensors: one at the front door and one at the back door. The sensors send a signal to the security system whenever someone enters the house by the first **or** the second door

Boolean algebra

From these basic operations, we can build other logic gates such as XOR or NAND

A	NOT A
0	1
1	0

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

Boolean algebra: how to sum?

You all know how to sum two number in decimal

Decimal

```
  1 1
  149
+  73
-----
  222
```

Binary

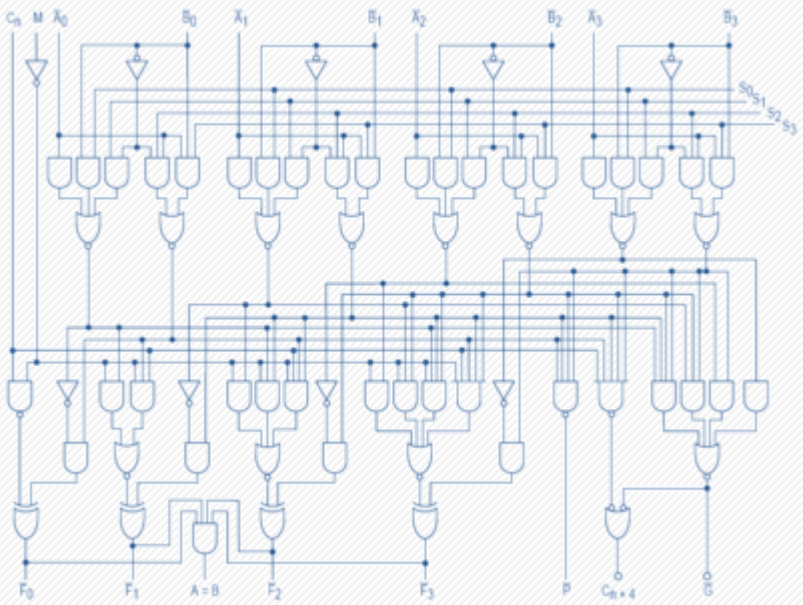
```
      1
  10010101
+ 01001001
----------
 11011110
```

→ 222 in decimal

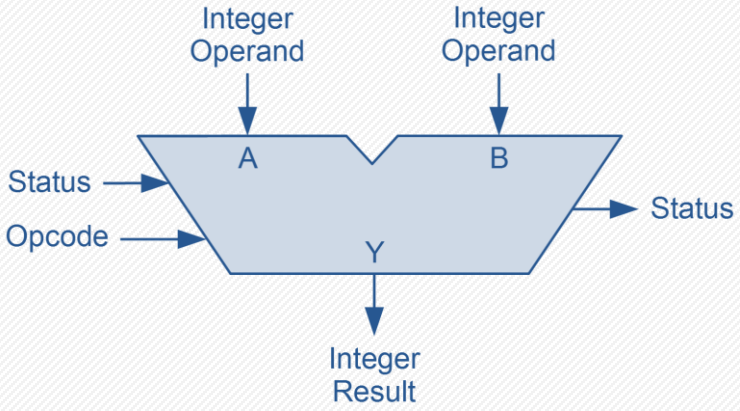
Any of the fundamental algebraic operations requires a combination of logic gates. We call these fundamental computing blocks: **ALU (Arithmetic & Logic Unit)**

ALU: Arithmetic and Logic Unit

Here is how an ALU looks like



Here is its symbol



Where Opcode is an input to control what operation you want to perform (mul, sum, etc.). Status give information such as the sign of the number, is it overflowed?

In a computer, ALUs do all the arithmetic and logic computations

Memory

Two types of memory

RAM: Random Access memory

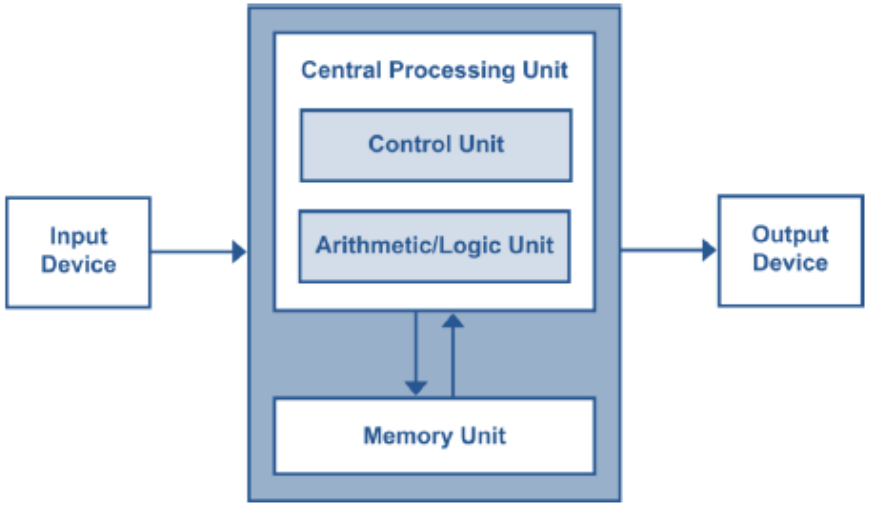
- Temporary storage of data and instructions for the CPU
- All the memory is erased when the power is OFF
- Build with logical gate (Latches)

Persistent storage

- Long term storage
- SSD, HDD, etc.

CPU

A CPU (Central Processing Unit) executes instructions stored in memory by performing arithmetic and logical operations on data.



The stored program approach used today is implemented using **von Neumann architecture**, named after U.S. mathematician John von Neumann

Modern computer architecture

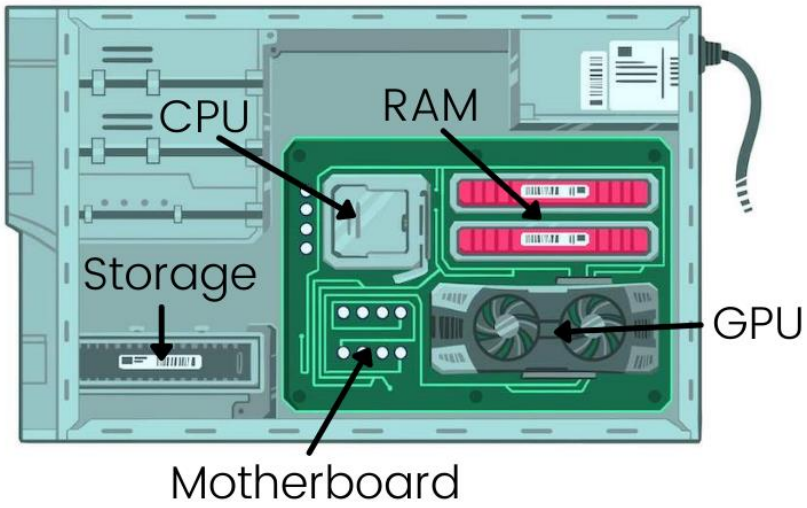
In modern computers (PCs), the major components in a von Neumann machine reside physically in a circuit board called the **motherboard**

- The CPU, memory, expansion cards, and other components are plugged into slots so they can be replaced
- Hard drives, CD drives, and other storage devices are connected to the motherboard through cables

The central processing unit (CPU) is the "brain" of the machine

- Its **arithmetic/logic unit** (ALU) performs millions or billions of calculations per second
- The **control unit** is the main organizing force of the computer and directs the operation of the ALU

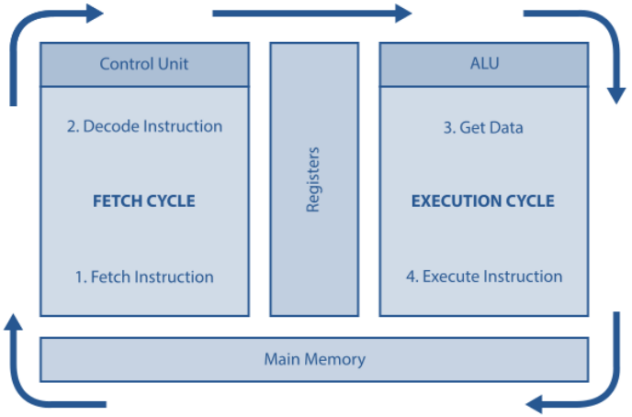
Modern computer architecture



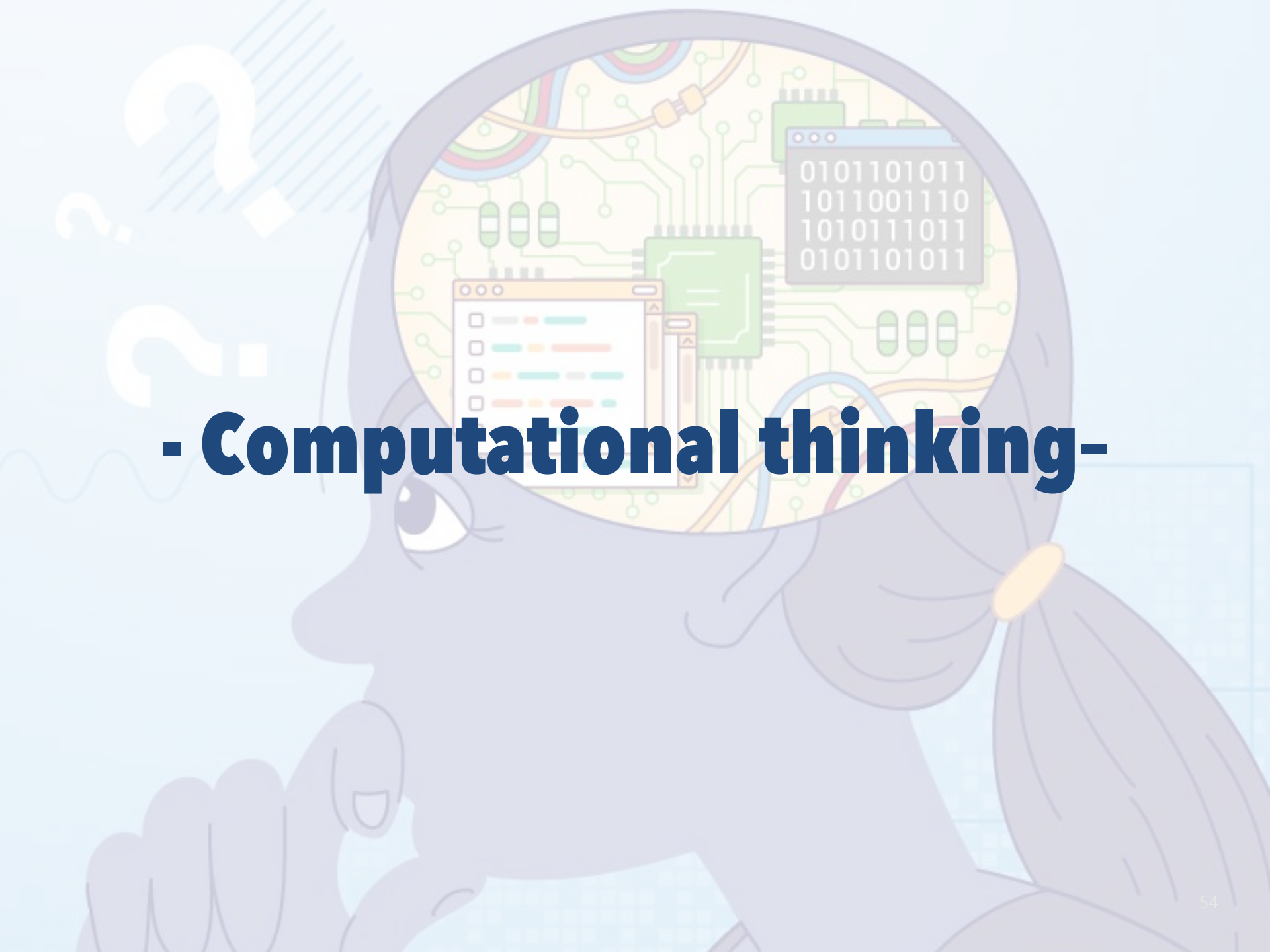
The fetch-decode-execute cycle

The system sends electrical signals that encode machine instructions and data

- The CPU **fetches** the instructions and data from memory as needed
- The control unit **decodes** each instruction to figure out what it is (an addition operation, subtraction, etc.)
- Data values (e.g., numbers to be added and their resultant sum) are stored temporarily in memory cells called **registers** within the CPU
- The ALU **executes** the instruction, saving the result in the registers and main memory



This whole process is known as the **fetch-decode-execute cycle**



- Computational thinking-

Computational Thinking

➔ How computer scientists think – how they reason and work through problems

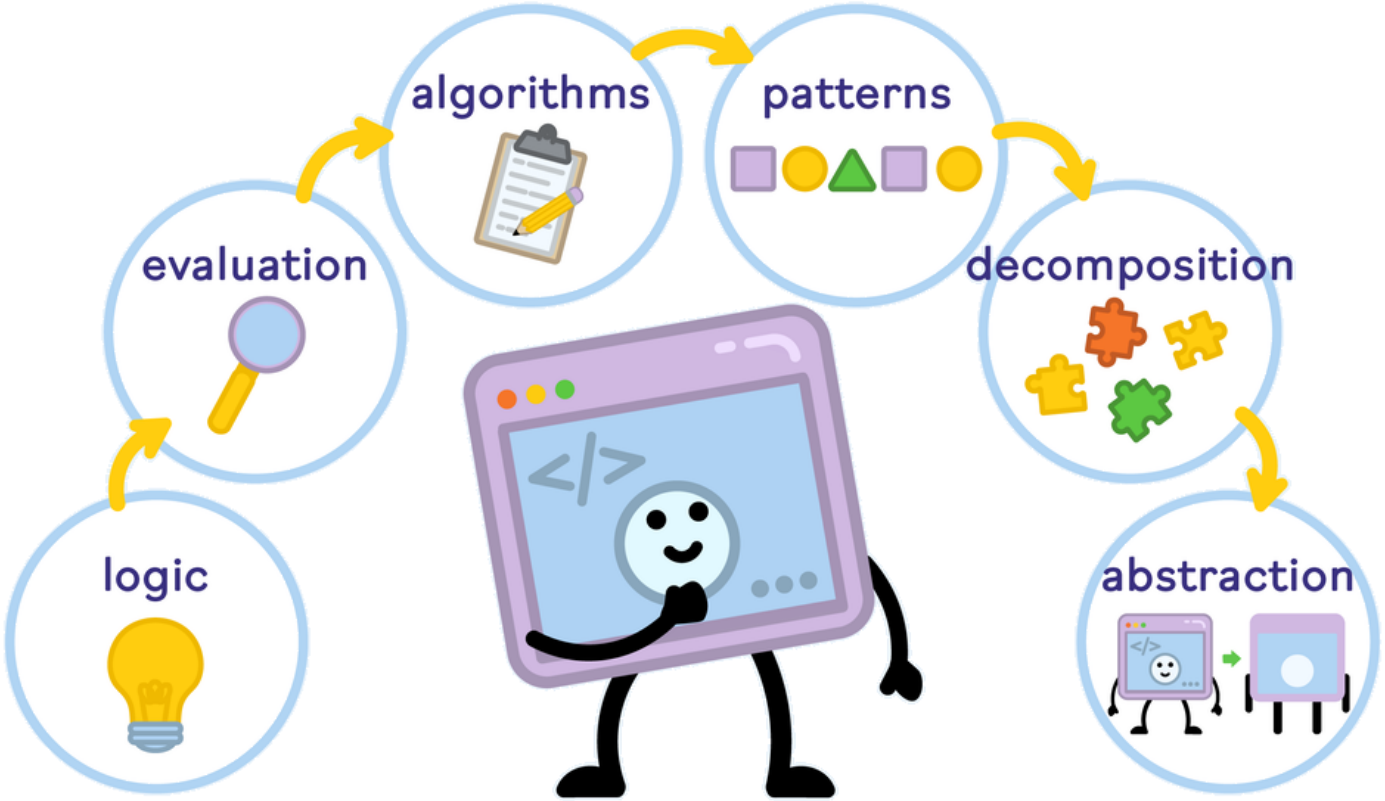
Computer science encompasses many sub-disciplines that support the goal of solving problems:

- Computer theory areas ➔ these are the heart and soul of computer science
  - Algorithms
  - Data structures
- Computer systems areas
  - Hardware design
  - Operating systems
  - Networks
- Computer software and applications
  - Software engineering
  - Programming languages
  - Databases
  - Artificial intelligence
  - Computer graphics

A major goal of this course is to help you develop your computational thinking and problem-solving skills

Computational Thinking

Computational thinking is a problem-solving method that involves breaking down complex problems into smaller, more manageable parts



Source: Computational Thinking — Starfire Education

Computational Thinking

Let's take the example of a farmer who wants to optimize his crop yields using computational thinking

1. Decomposition

The farmer breaks down the problem of optimizing his crop yields into smaller, more manageable parts. These parts include soil quality, irrigation, pest control, crop rotation, and harvesting.

2. Pattern recognition

The farmer identifies patterns in his data, such as which crops perform best in different types of soil, which pests are most common, and which irrigation techniques are most effective.

3. Abstraction

The farmer focuses on the essential details of the problem, such as the factors that influence crop yields, and ignores irrelevant information, such as weather patterns that cannot be controlled.

4. Algorithm design

The farmer designs a step-by-step plan for optimizing his crop yields based on his research and analysis. He might create a schedule for crop rotation that minimizes soil depletion, develop a pest control plan that uses natural predators, and use data analysis to find effective irrigation techniques.

5. Evaluation

The farmer evaluates the effectiveness and efficiency of his solutions by measuring his crop yields over time and comparing them to previous years. He adjusts his approach as needed based on the results.

Computational Thinking

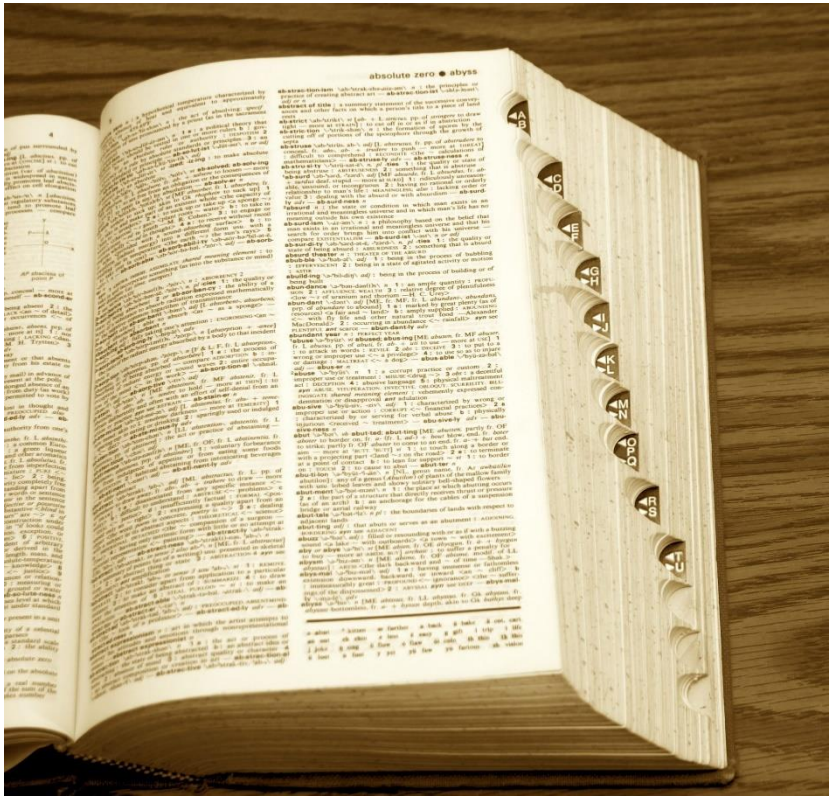
Electronic health records are very important

- Consider issues (technical + others) that arise in providing a system to medical professionals and others who need access to digital medical records:
  - What data will be stored? How? In what format?
  - How will the data be accessed and displayed?
  - Who will have access? How will the data be secured?
  - How will the data be backed up and preserved?

- Answering these questions requires **computational thinking**

Many ways of solving problems

Imagine you are looking for the word “Science” in a dictionary what algorithm can you design to find this word inside?



One solution would be to look at every single word from page 1 to page N. And you stop when you find the correct word.

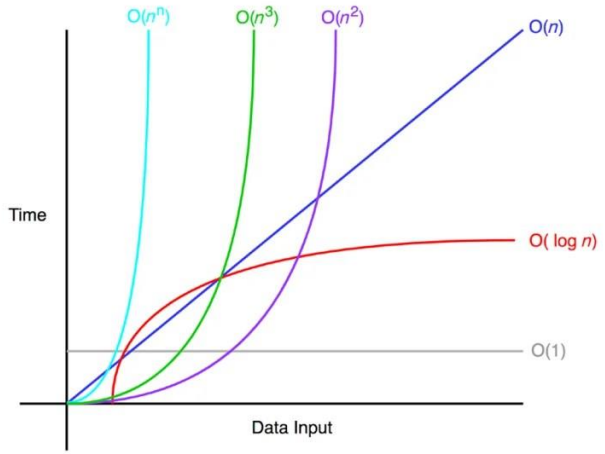
What can you say about it?

- Will it work?
- Is it effective?

Many ways of solving problems

Instead you can opt for a **binary search** strategy

1. You select the word at the middle of the dictionary
2. If the word you are looking for is located before this word, then you only need to check your word in the first half of the dictionnary
3. Else you only need to look into the second half of the dictionnary
4. Reiterate the operations 1,2,3 on the remaining part of the dictionnary until you find your word

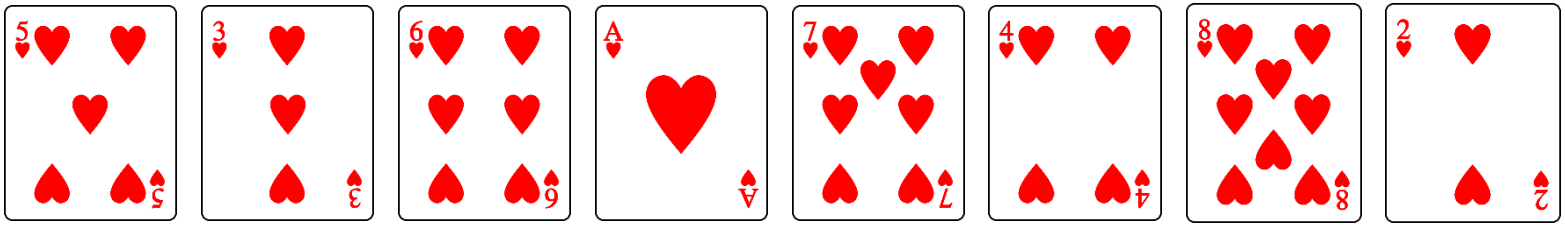


A classical problem: sorting

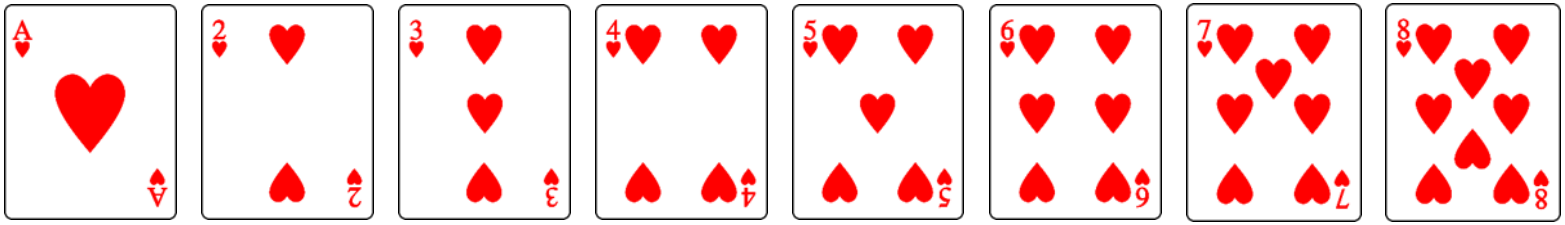
Sorting: an important problem – arises frequently in computer science

- Suppose there is a deck of cards to be put in order
- Example: We have the Ace up to 8 of Hearts

Given this:

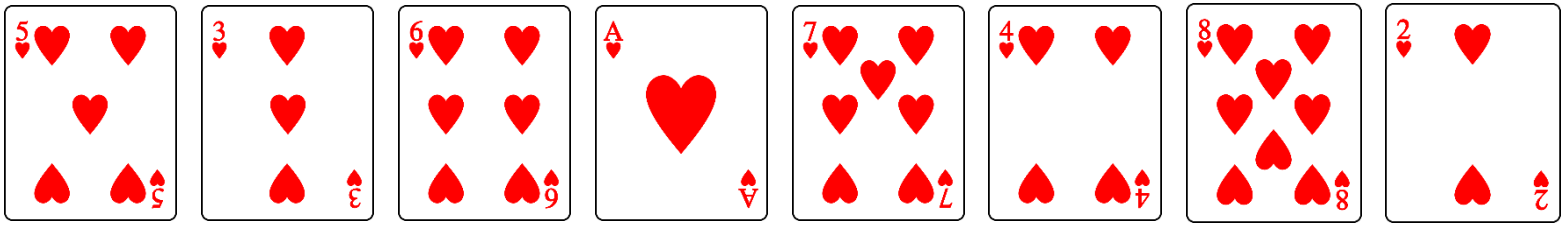


You want to obtain the following:



A classical problem: sorting

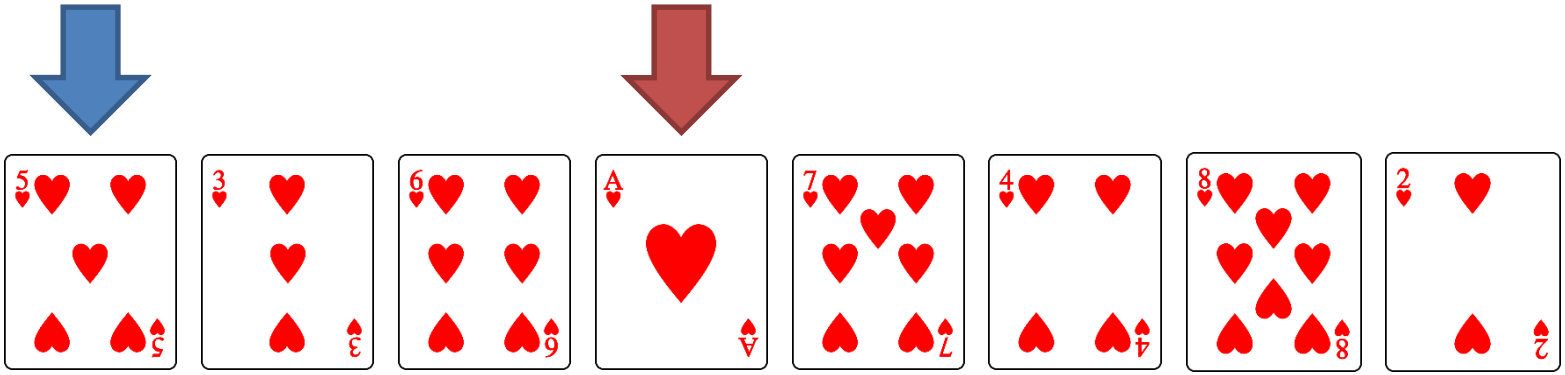
Now you want to explain a little child to solve this problem step by step such that it always works.
What steps would you give?



A classical problem: sorting

One technique called **Selection sort**

It repeatedly searches for and swaps cards in the list

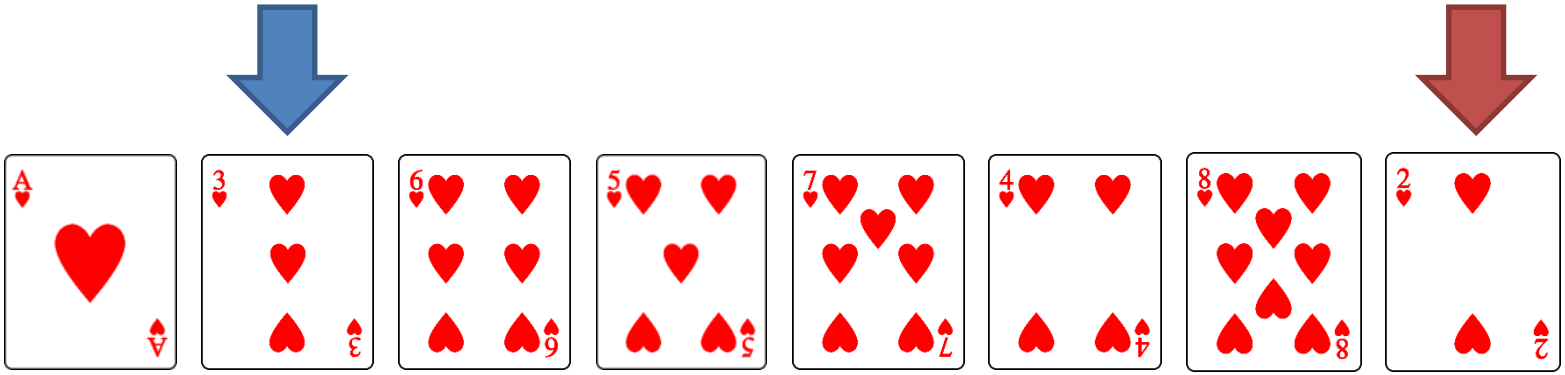


1. First, find the smallest item and exchange it with the card in the first position

A classical problem: sorting

One technique called **Selection sort**

It repeatedly searches for and swaps cards in the list

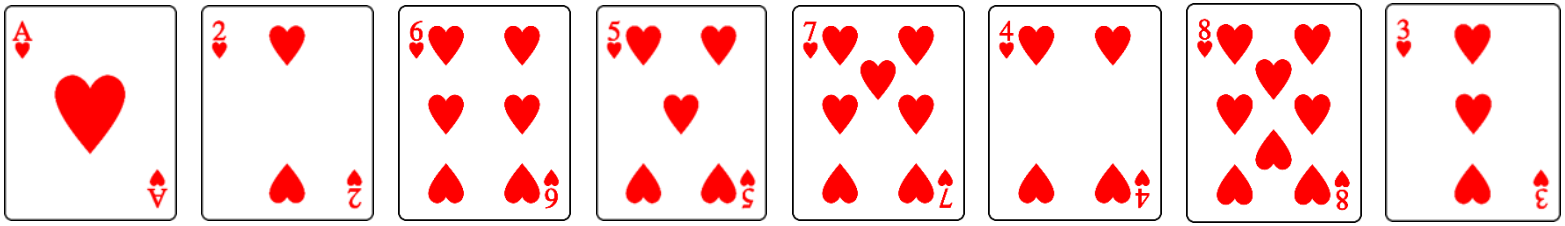


1. First, find the smallest item and exchange it with the card in the first position
2. Select the second smallest item and exchange it with the card in the second position

A classical problem: sorting

One technique called **Selection sort**

It repeatedly searches for and swaps cards in the list

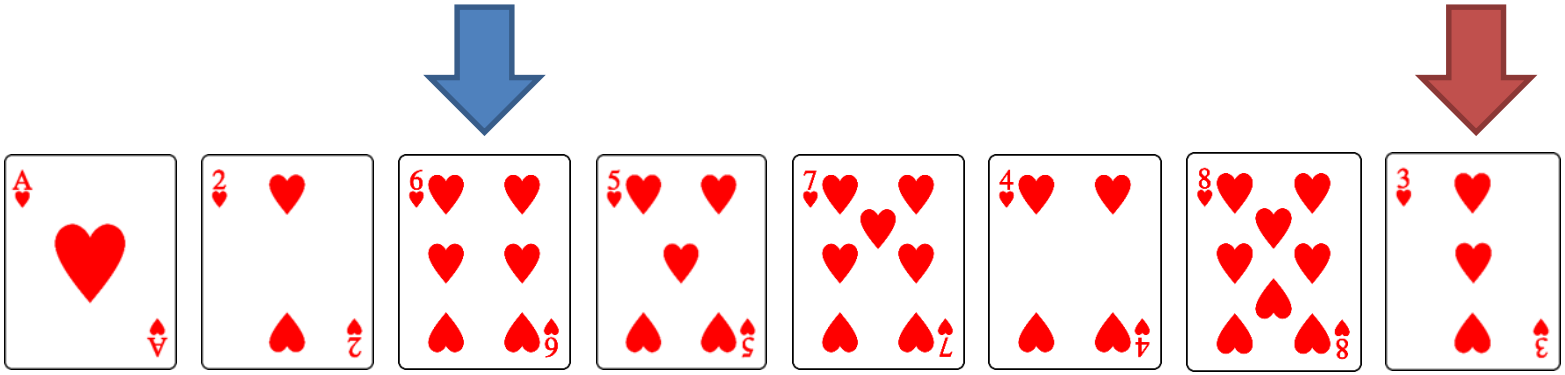


1. First, find the smallest item and exchange it with the card in the first position
2. Select the second smallest item and exchange it with the card in the second position

A classical problem: sorting

One technique called **Selection sort**

It repeatedly searches for and swaps cards in the list

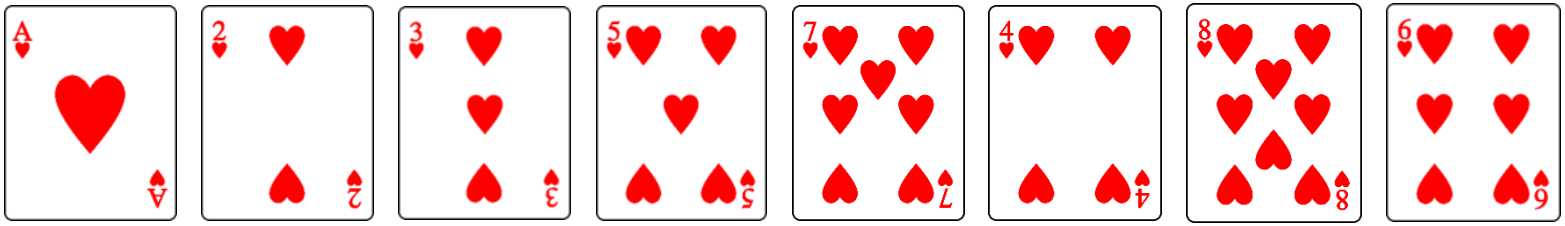


1. First, find the smallest item and exchange it with the card in the first position
2. Select the second smallest item and exchange it with the card in the second position
3. Select the third smallest item and exchange it with the card in the third position

A classical problem: sorting

One technique called **Selection sort**

It repeatedly searches for and swaps cards in the list

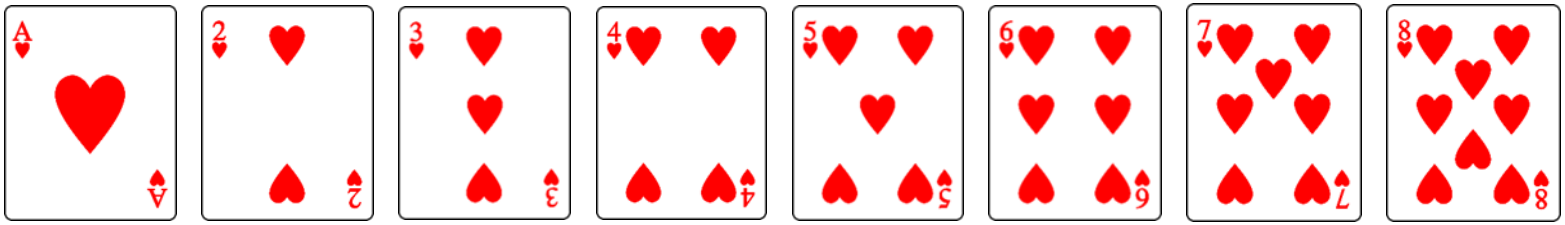


1. First, find the smallest item and exchange it with the card in the first position
2. Select the second smallest item and exchange it with the card in the second position
3. Select the third smallest item and exchange it with the card in the third position

A classical problem: sorting

One technique called **Selection sort**

It repeatedly searches for and swaps cards in the list



1. First, find the smallest item and exchange it with the card in the first position
2. Select the second smallest item and exchange it with the card in the second position
3. Select the third smallest item and exchange it with the card in the third position
4. Keep going!

A classical problem: sorting

One technique called **Selection sort**
It repeatedly searches for and swaps cards in the list

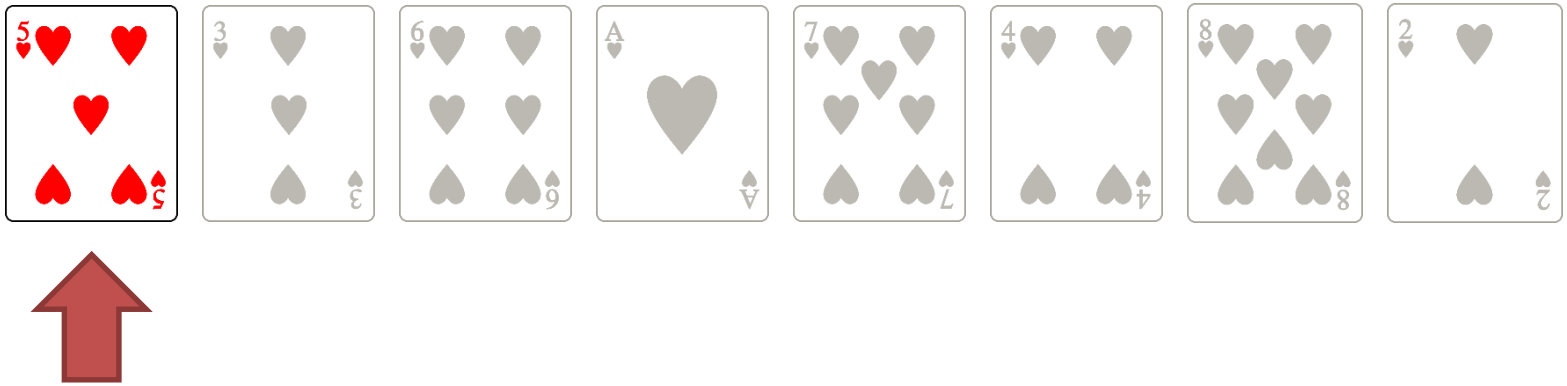
What do you think about Selection sort? Is it a good technique?

$$O(n^2)$$

A classical problem: sorting

Let's see another technique called **Insertion sort**

It repeatedly inserts the "next" card into its correct spot

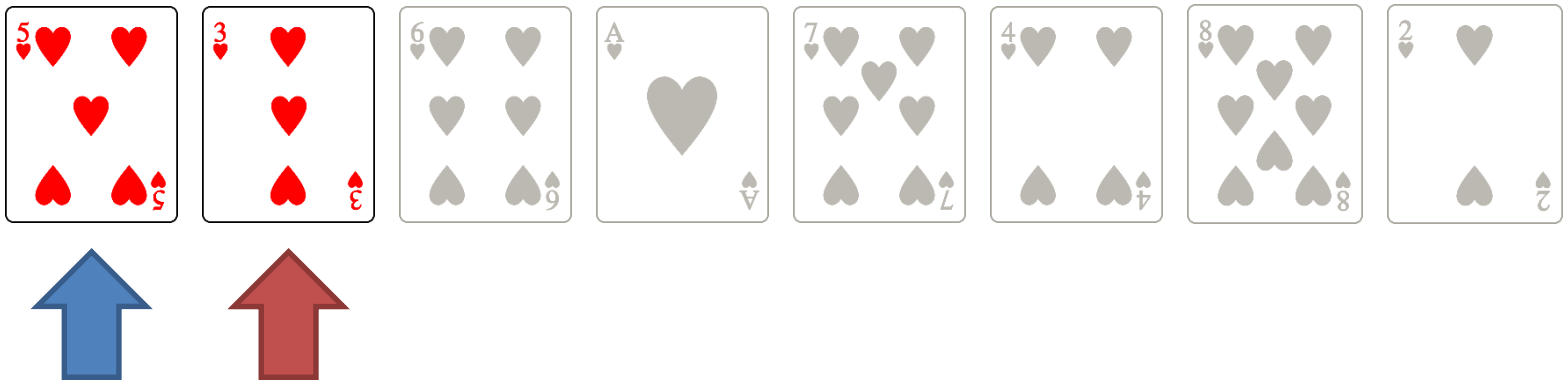


```
The first element has no left neighbor so will
start directly by the second element
```

A classical problem: sorting

Let's see another technique called **Insertion sort**

It repeatedly inserts the "next" card into its correct spot

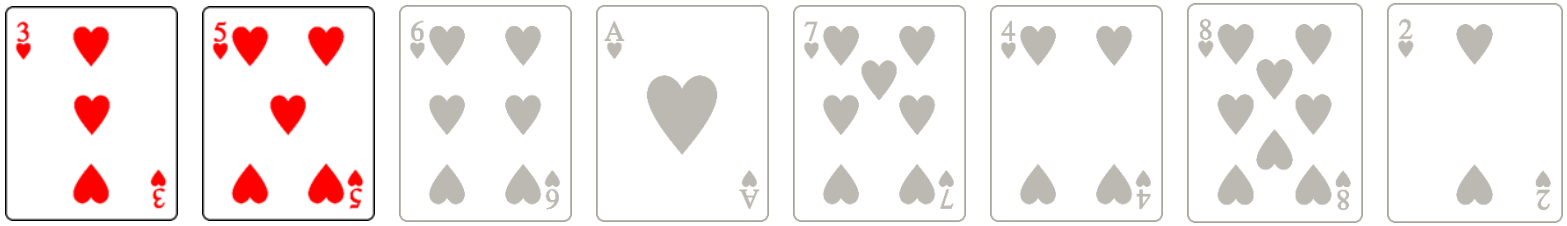


The second element is smaller than its left neighbor → we swap them!

A classical problem: sorting

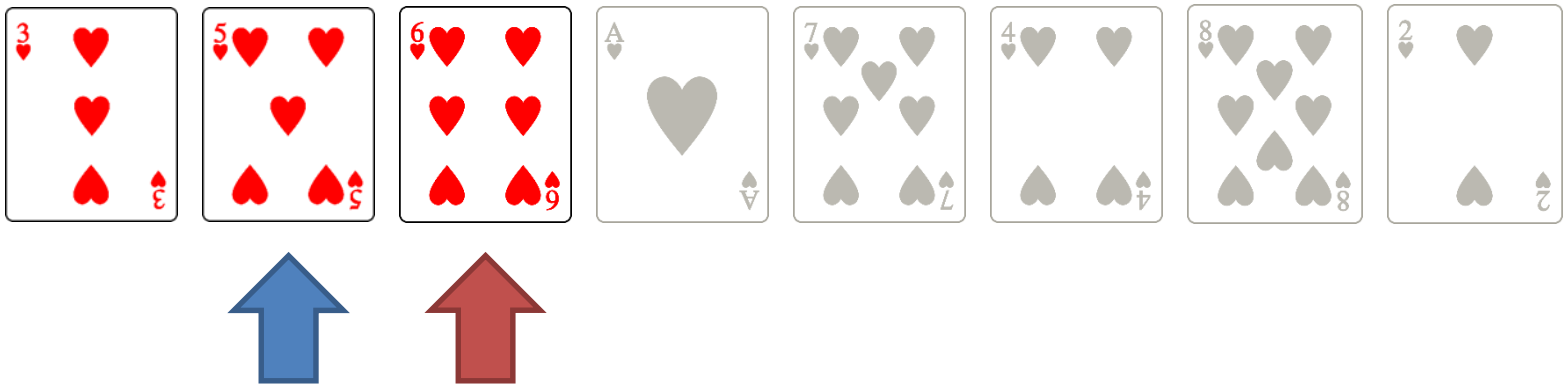
Let's see another technique called **Insertion sort**

It repeatedly inserts the "next" card into its correct spot



A classical problem: sorting

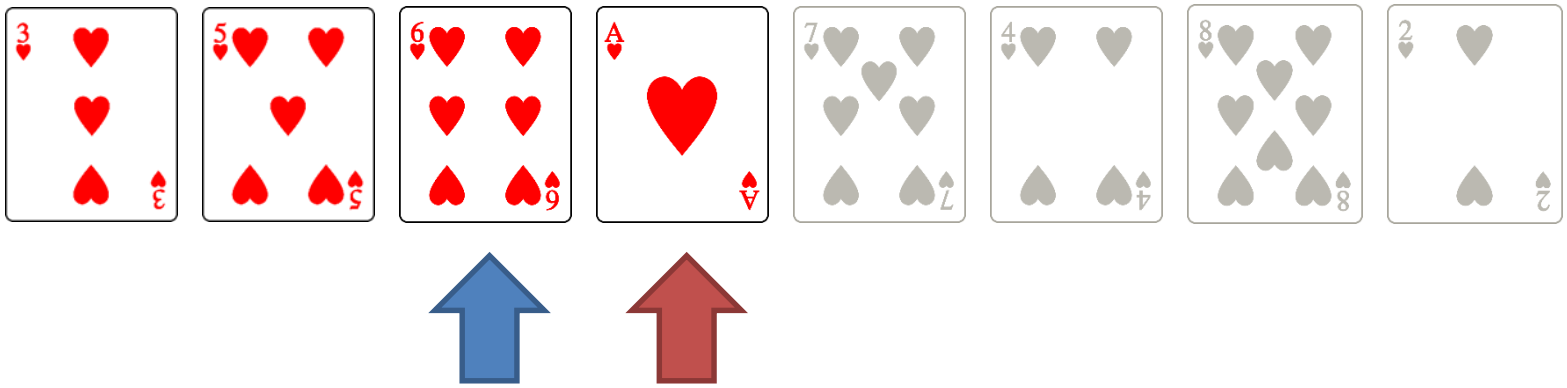
Let's see another technique called **Insertion sort**
It repeatedly inserts the "next" card into its correct spot



The third element is bigger than its left neighbor
→ we **do not** swap them!

A classical problem: sorting

Let's see another technique called **Insertion sort**
It repeatedly inserts the "next" card into its correct spot

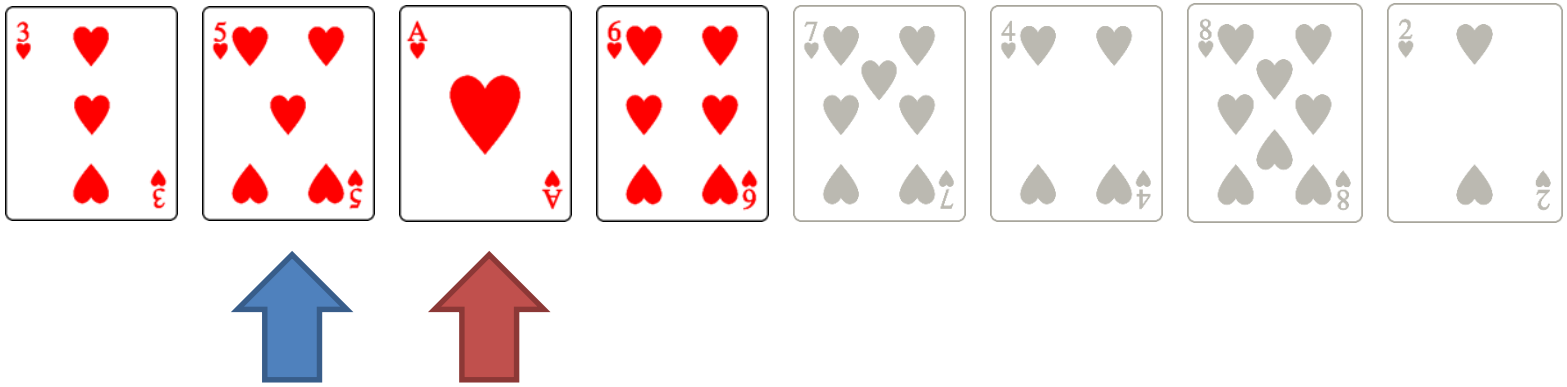


The fourth element is smaller than its left neighbor → we swap them!

A classical problem: sorting

Let's see another technique called **Insertion sort**

It repeatedly inserts the "next" card into its correct spot

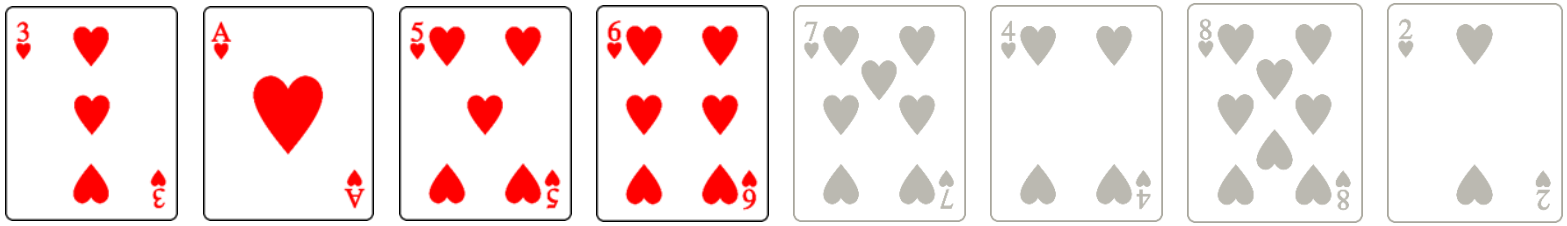


Continue the insertion process until all this part is sorted

A classical problem: sorting

Let's see another technique called **Insertion sort**

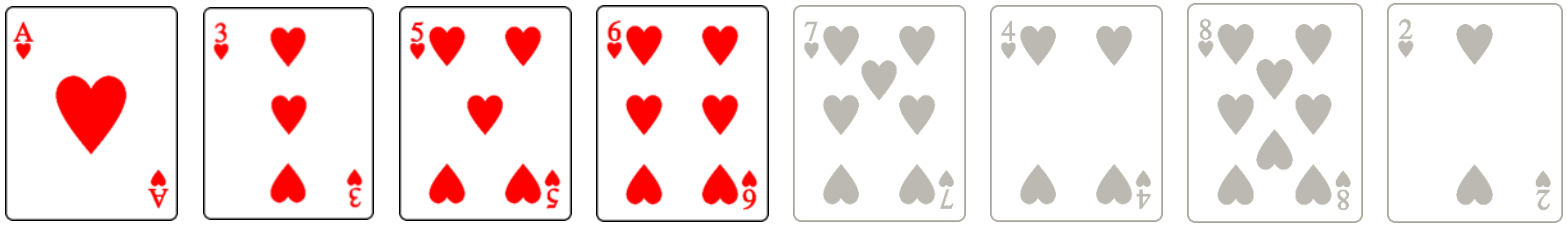
It repeatedly inserts the "next" card into its correct spot



A classical problem: sorting

Let's see another technique called **Insertion sort**

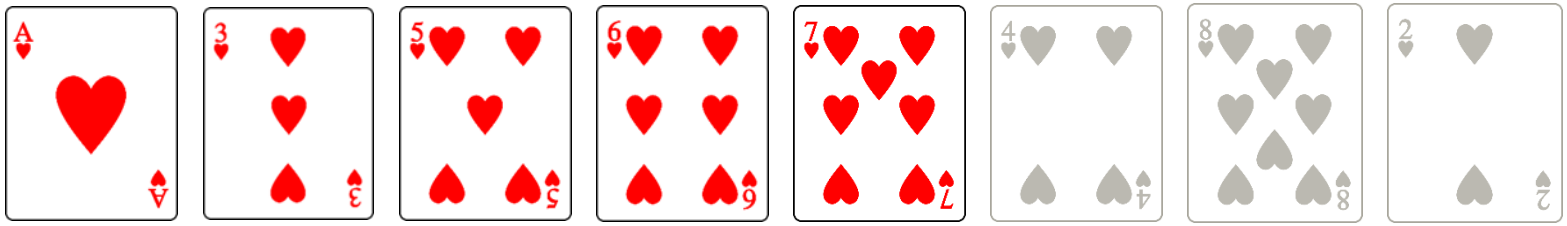
It repeatedly inserts the "next" card into its correct spot



A classical problem: sorting

Let's see another technique called **Insertion sort**

It repeatedly inserts the "next" card into its correct spot

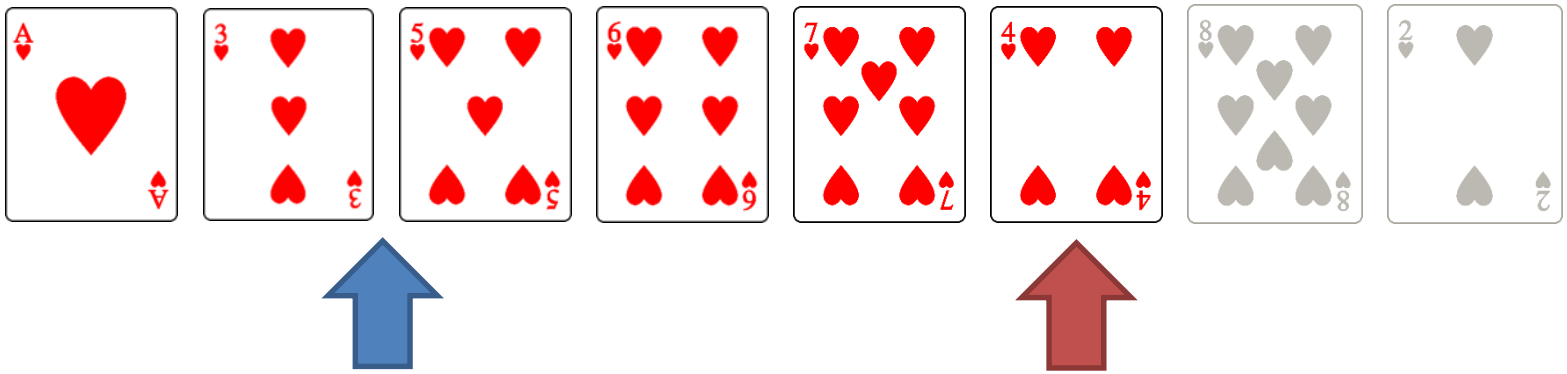


We move to the next element

A classical problem: sorting

Let's see another technique called **Insertion sort**

It repeatedly inserts the "next" card into its correct spot

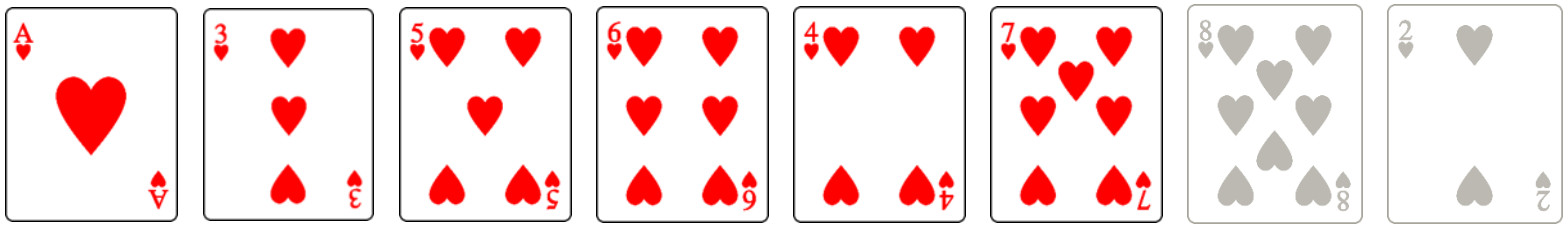


And we will insert the element at the right place by successively comparing it with its left neighbor

A classical problem: sorting

Let's see another technique called **Insertion sort**

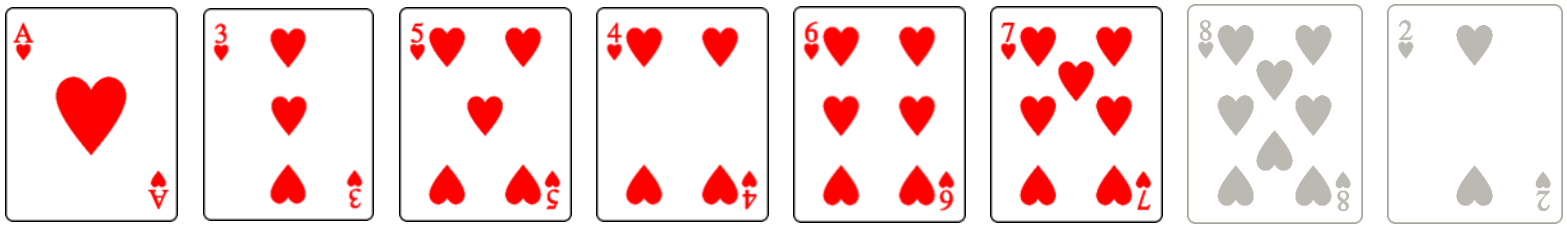
It repeatedly inserts the "next" card into its correct spot



A classical problem: sorting

Let's see another technique called **Insertion sort**

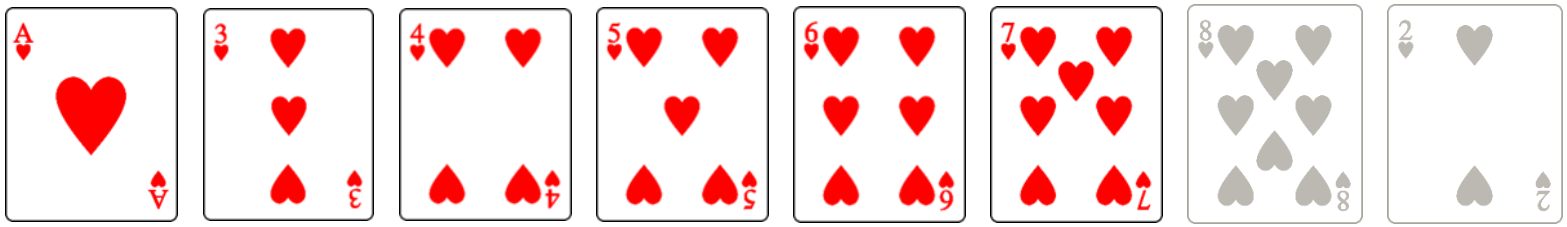
It repeatedly inserts the "next" card into its correct spot



A classical problem: sorting

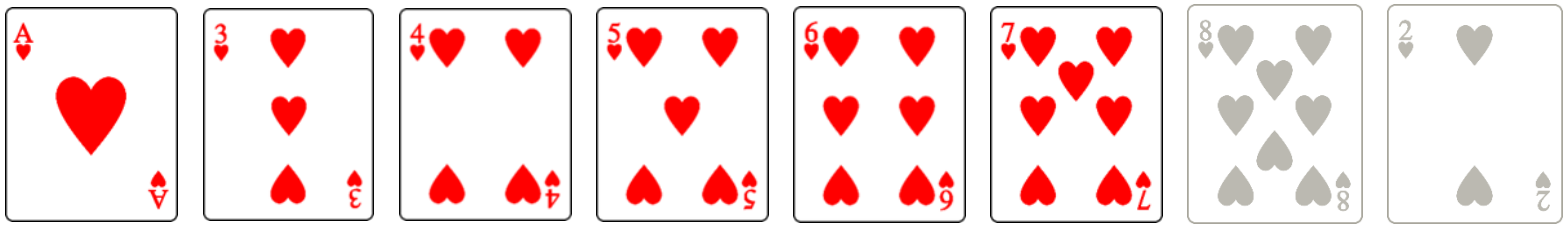
Let's see another technique called **Insertion sort**

It repeatedly inserts the "next" card into its correct spot



A classical problem: sorting

Let's see another technique called **Insertion sort**
It repeatedly inserts the "next" card into its correct spot



Continue the insertion process until you reach the end of the list

```
for i : 1 to length(A) -1
   j = i
   while j > 0 and A[j-1] > A[j]
      swap A[j] and A[j-1]
      j = j - 1
```

A classical problem: sorting

- We saw there can be different ways to solve the same computational problem
  - ➔ can derive many different algorithms for sorting
- Algorithm:
  - A set of concrete steps
  - Steps solve a problem or accomplish some task
  - Solve in a finite amount of time
- The **Selection Sort** and **Insertion Sort** algorithms are only two ways of sorting a list of values
- New problem: You want to sort a list of student records by their GPAs.
  - Would both of these algorithms work?
  - Yes! A sign of a good algorithm is that it is **general** ➔ can solve a variety of similar problems

A classical problem: sorting

Limits of computing

What a computer can do:	What a computer cannot do:
Send email to a person if email address is known.	Find email of a person we met at a coffee shop.
Calculate different investment options based on historical data.	Choose a perfect investment or predict the future of companies.
Find information about colleges offering computer science courses.	Make a perfect decision on the best school to attend.
Solve well defined problems.	Solve ambiguous problems.

Limits of computing

- If a computer tries to analyze every possible sequence of moves in response to this opening in a game of chess, it will have to consider over 10^{43} games.
- If a computer could solve one trillion combinations per second, it will compute the perfect game of chess if we are patient enough to wait 10^{21} years.

To sum up

- Computer science is the discipline of how to solve problems using computers
- We strive for efficient, general solutions that will work on a wide variety of problems
- Although computer science has existed as a field for about 70 years, its roots in mathematics and computation go back thousands of years!
- CS is a field that relies partly on old mathematical ideas but experiences advances in development of new techniques at an extraordinary pace
- This semester, you will be exposed to many of the modern topics in CS and some of the older mathematical content that is still very relevant today

Sources and references

Acknowledgement:
These slides are revised versions of slides prepared by Prof. Arthur Lee, Tony Mione, Pravin Pawar, and Alex Kuhn for earlier CSE 101 classes. Some slides are based on Prof. Kevin McDonald at SBU CSE 101 lecture notes and the textbook by John Conery.

Additional resources from which I strongly relied for preparing this class:
Early Computing: Crash Course Computer Science #1 – YouTube
CS50 2019 - Lecture 0 - Computational Thinking, Scratch - YouTube