ICDS Spring 2025

Course overview

Persons to know

- **Instructor:** Zhaonan Wang

- **Teaching Assistants:**
  - Sven Simikin (Sec.1), Yiyao Yang, Jennifer Feng (Sec.2)

- **Learning Assistants by ARC:**
  - Steven Zhang, Mohamed Hendy

Course structure

Seg 1: computer architecture with a history	
Computer architecture/history	Python review Quiz 1
Seg2: algorithmic thinking and practice	
Algorithm/complexity families	Python: OOP Midterm
Seg 3: data science intro	
Machine learning/A.I.	Chat system Quiz 2
Seg 4: advanced topics	
Cybersecurity, networking, etc.	Final project Final

Learning Goals

After taking this course, you are expected to

- Know the fundamentals of modern computer science
    - Hardware settings and algorithm basics
- Master Python and OOP
    - syntax, and ways to design programs
- Get a general view about data science
    - the tasks it tries to solve and tools it uses for solving problems

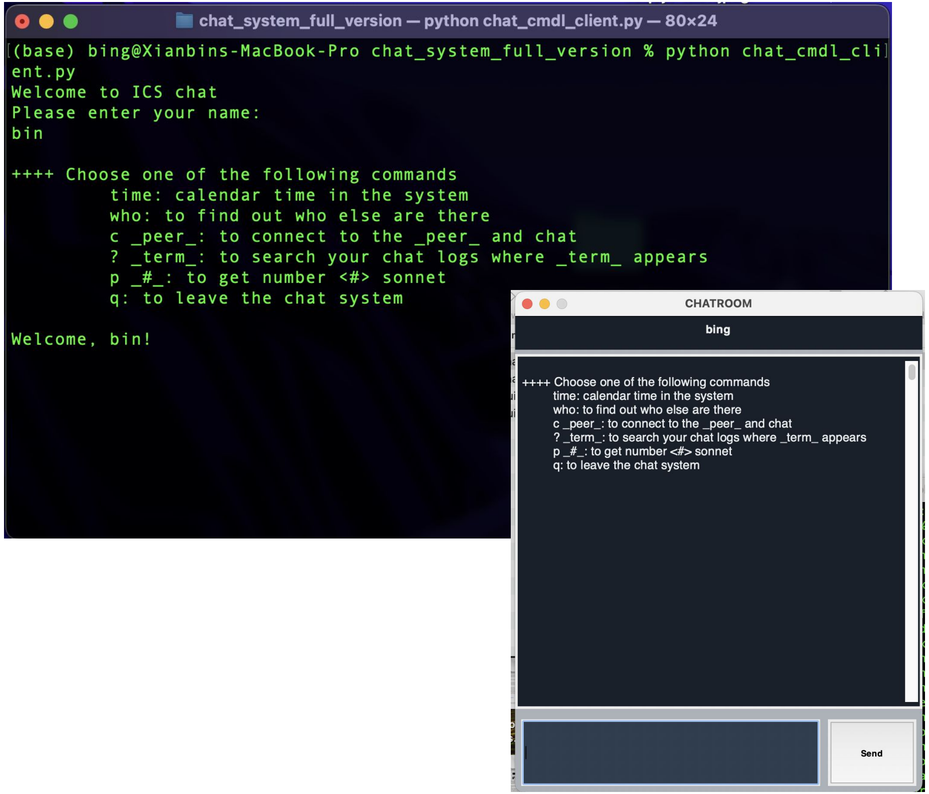
How to study this course

- **Lectures:**
  - Introductions to principles, theories, and approaches in computer science and data science.
- **Recitations:**
  - crash courses of programming + in-class exercises
- **Assignments**:
  - Programming exercises for you to keep involved!
- **Office hours:**
  - If you have any problems in assignments and exercises, please meet with us.
- **GenAI tools:**
  - Use it responsibly for in-class exercises and assignments, but never in quizzes and exams.

Assignments

- Four programming assignments in Week 3 to 7
  - You will have one week to work on each one
  - Submit to Gradescope
- Three project assignments (Unit Projects) in Week 8 to 10
  - Submit to Gradescope
- Auto-graders will be used: you can use its feedback to improve your solutions
- Final project in Week 12 to 14
  - You will team up and extend the Unit Projects you develop
  - You will record a video to present your project

The Chat System

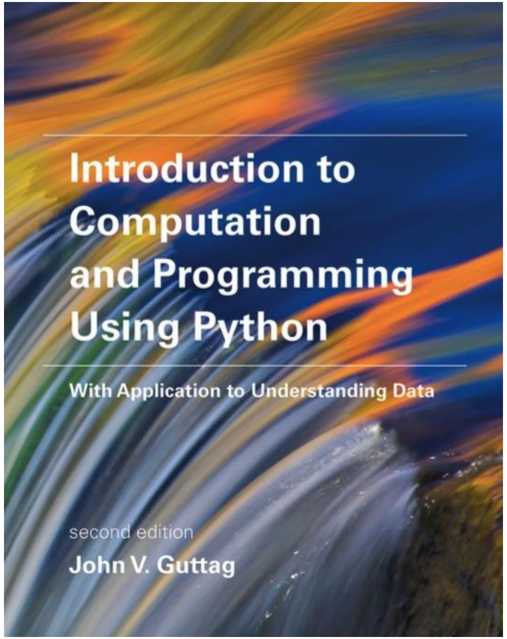

chat_system_full_version — python chat_cmdl_client.py — 80×24

```
(base) bing@Xianbins-MacBook-Pro chat_system_full_version % python chat_cmdl_cli
ent.py
Welcome to ICS chat
Please enter your name:
bin

++++ Choose one of the following commands
         time: calendar time in the system
         who: to find out who else are there
         c _peer_: to connect to the _peer_ and chat
         ? _term_: to search your chat logs where _term_ appears
         p _#_: to get number <#> sonnet
         q: to leave the chat system

Welcome, bin!
```

```
++++ Choose one of the following commands
        time: calendar time in the system
        who: to find out who else are there
        c _peer_: to connect to the _peer_ and chat
        ? _term_: to search your chat logs where _term_ appears
        p _#_: to get number <#> sonnet
        q: to leave the chat system
```

- A simplified WeChat
- Running in the terminal
- Users can log into the system and send messages to others.
- You will extend it to a GUI based chat system in the final project.

Reference

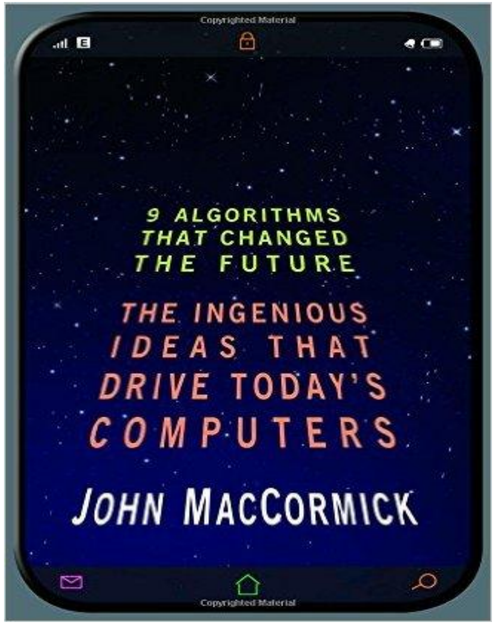


Introduction to Computation and Programming Using Python

With Application to Understanding Data

second edition

John V. Guttag



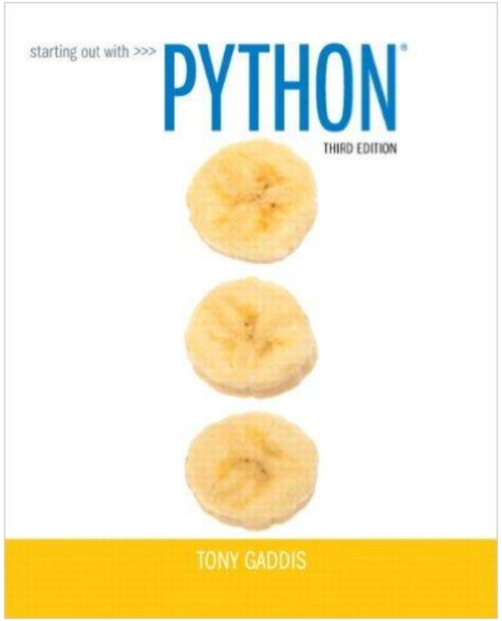
Copyrighted Material

9 ALGORITHMS THAT CHANGED THE FUTURE

THE INGENIOUS IDEAS THAT DRIVE TODAY'S COMPUTERS

JOHN MACCORMICK

Copyrighted Material



starting out with >>> PYTHON®

THIRD EDITION

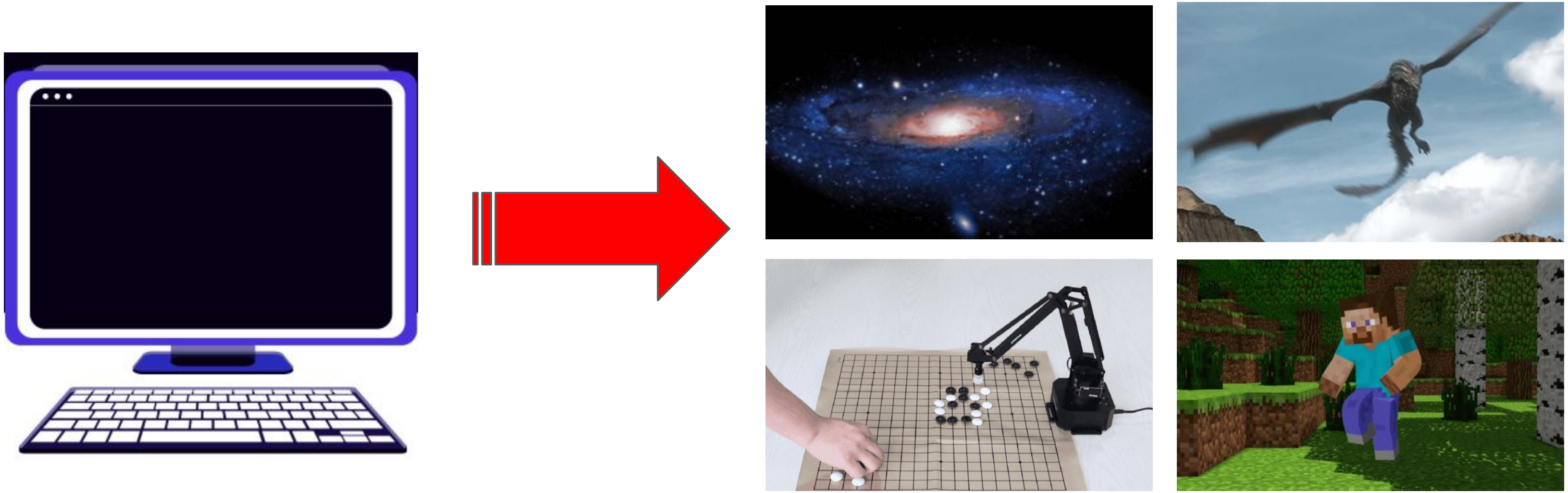
TONY GADDIS

ICDS Spring 2025

From Zero To One

The history and Boolean algebra

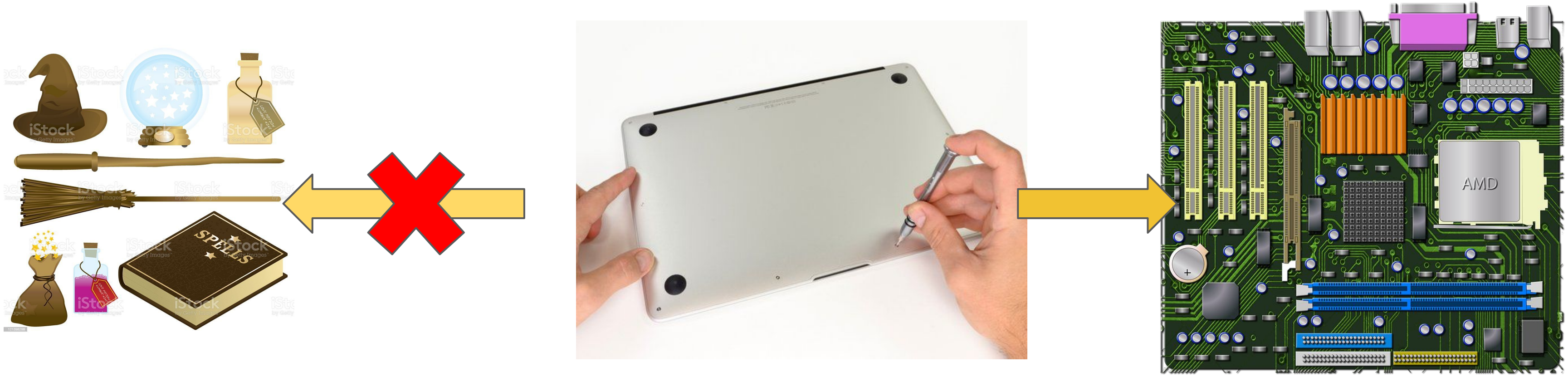
A computer is a wonder machine



- “Computer” comes from the word “compute” which means to calculate. ⇒ It seems to be a device to handle numbers.
- But it actually shows us many other fancy stuffs rather than numbers ⇒ How can it do this?

What’s inside a computer?

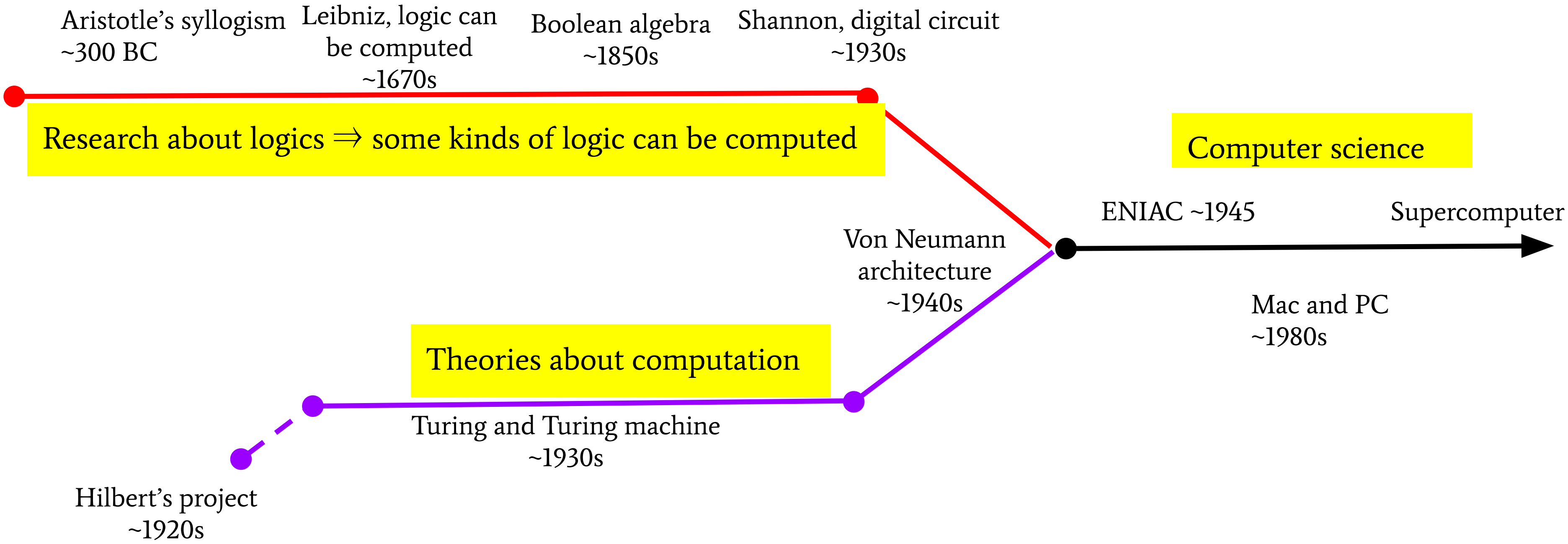
- Not a crystal ball/magic stuffs
- Inside a computer → a Printed Circuit Board (PCB)
    - circuits that can compute ⇒ doing logic inference like humans



Agenda

- **The history of computer science**
  - **The origins**
  - **Turing machine**
- Boolean algebra
  - Variables, operators, and truth table
  - Numbers in boolean algebra
- Appendix
  - Notations for integers and fractions

A history (tale) of computer science



Aristotle's syllogism ~300 BC

Leibniz, logic can be computed ~1670s

Boolean algebra ~1850s

Shannon, digital circuit ~1930s

Research about logics ⇒ some kinds of logic can be computed

Computer science

ENIAC ~1945

Supercomputer

Von Neumann architecture ~1940s

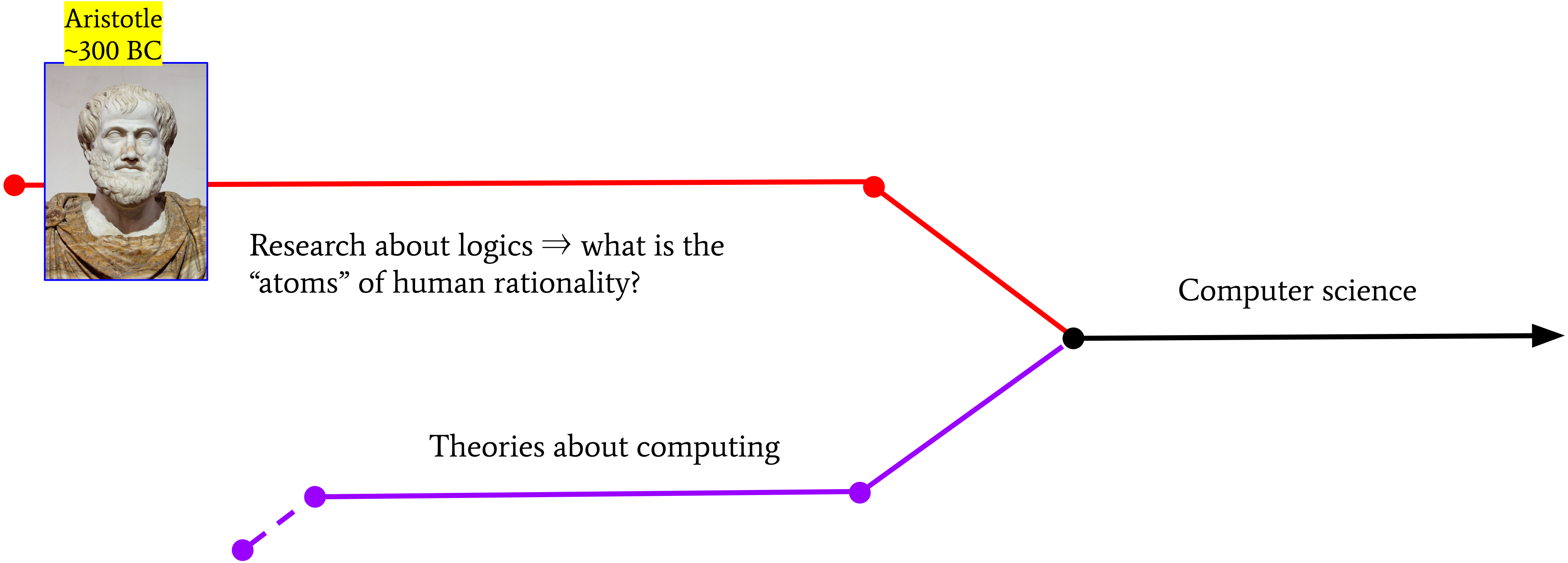
Mac and PC ~1980s

Theories about computation

Turing and Turing machine ~1930s

Hilbert's project ~1920s

Ancient Greeks' logics



Aristotle
~300 BC

Research about logics ⇒ what is the "atoms" of human rationality?

Theories about computing

Computer science

Ancient Greeks: formalized statements ⇒ logic

A famous form from Aristotle (syllogism):

- All men are mortal. (major premise)
- Socrates is a man. (minor premise)
- Hence, Socrates is mortal. (conclusion)

Another Example:

- If a is an even number, then, a · b is an even number.
- a = 6
- Therefore, 6 · b is even.

Logical Forms

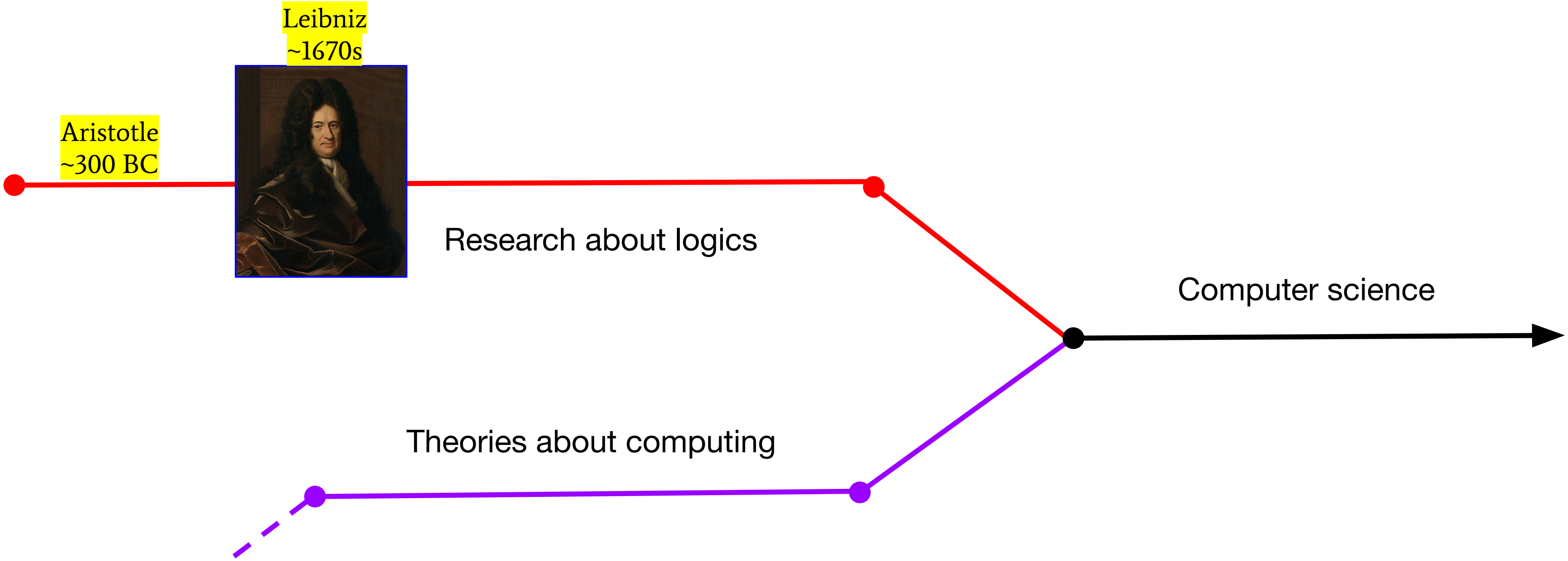
List of valid argument forms - Wikipedia.

The previous two examples have the same form (Modus ponens) but with different values for p and q.

- Modus ponens:
  If (p), then (q) [if you are a man (p), then, you are mortal (q)]
  (p) [Socrates is a man (p)]
  Therefore (q) [Socrates is mortal (q)]

- Logic is a set of **rules** that we have in our minds. It represents how we do reasoning.
- Logic is independent to its carriers. (There are many carriers, e.g., English, French, Chinese, …)

A tale of computer science



Leibniz
~1670s

Aristotle
~300 BC

Research about logics

Computer science

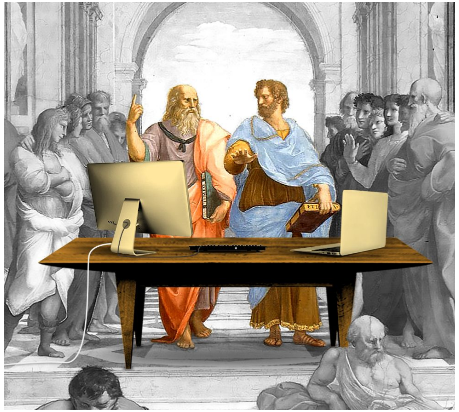
Theories about computing

Leibniz’s idea: logics can be computed

Gottfried Wilhelm von Leibniz, 1646 - 1716

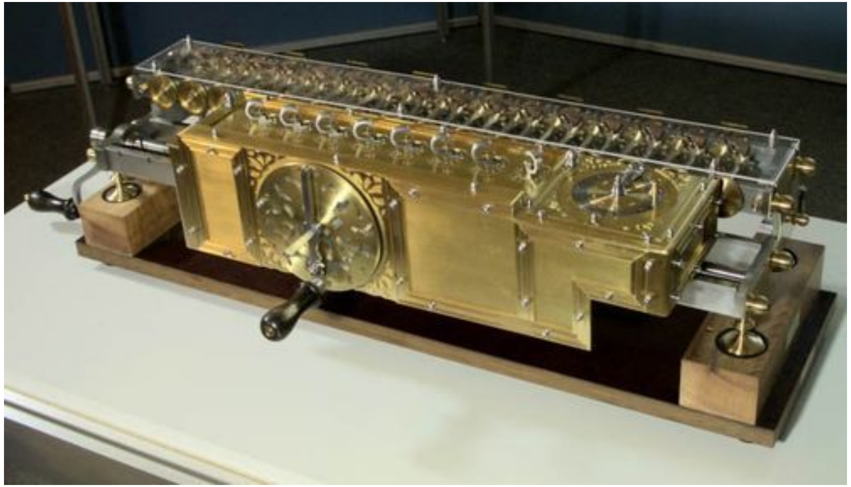
“If controversies were to arise, there would be no more need for disputation between two philosophers than two accountants. For it would suffice to take their pencils in their hands, to sit down to their slates, and to say to each other (with a friend as a witness if they like): Let us calculate.”

--- Leibniz, in Russell’s translation (1937)



Leibniz’s Stepped Reckoner

Logics ⇔ Symbolic language (e.g., math) ⇔ Physical device ?

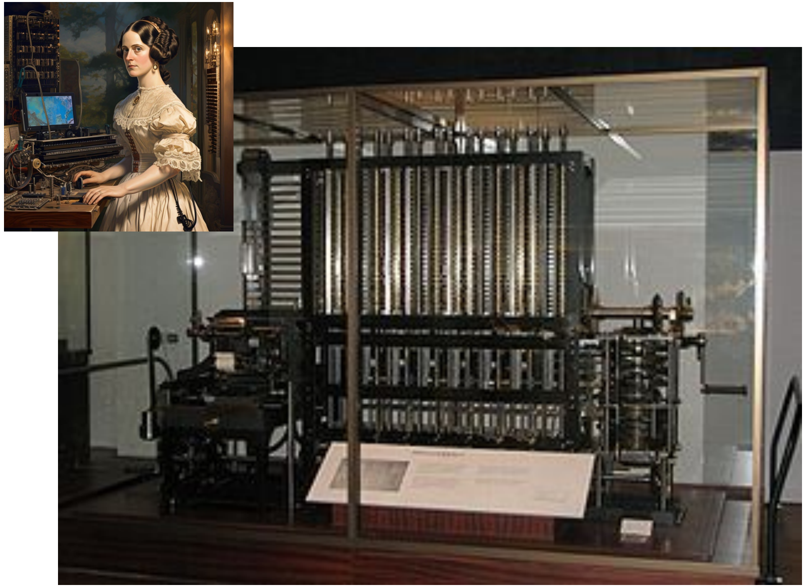


Stepped Reckoner (Wikipedia)

In 1670s, Leibniz built a device called Stepped Reckoner, the first digital mechanical calculator in the world.

- It proves Leibniz’s idea to some extent ⇒ the Leibniz wheel, is virtually a physical representation of some logic
- unfortunately, it failed to work reliably due to a design flaw in the carry mechanism

Charles Babbage's difference engine

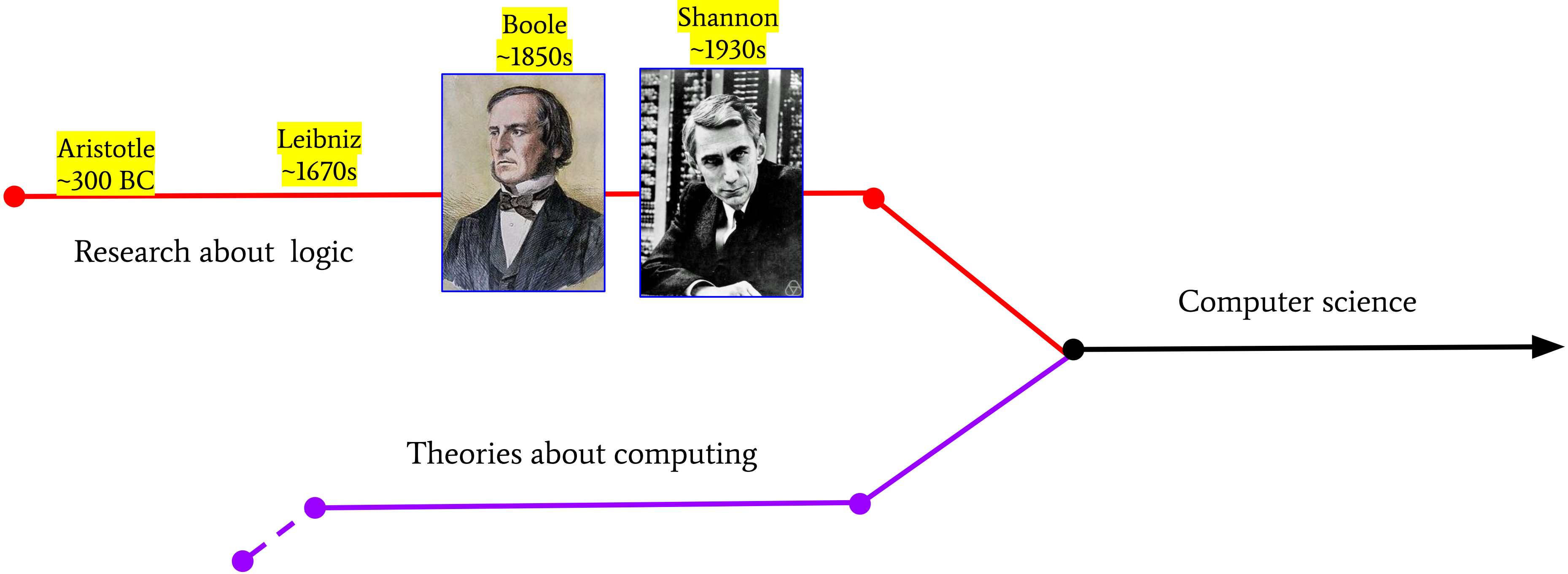


Difference machine is a calculating machine designed in the 1820s, was created by Charles Babbage

Well designed, but ⇒ Problems:

- Using decimal number system ⇒ 10 digits ⇒ 10 types of cogs ⇒ Failed due to the low metalworking technique of that era
  - Precision of the gears is not satisfying
  - Errors accumulated among gears
- the algebra language at that time was not good for making computing devices

A tale of computer science



Boole
~1850s

Shannon
~1930s

Aristotle
~300 BC

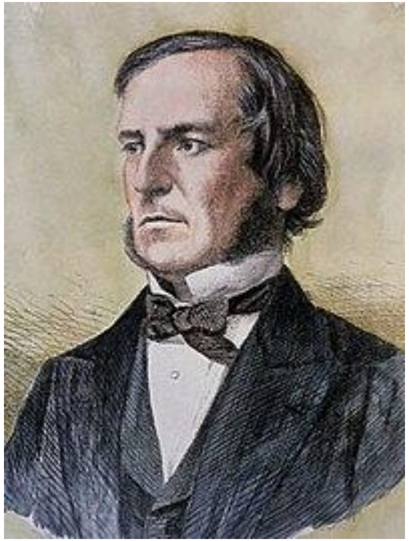
Leibniz
~1670s

Research about logic

Computer science

Theories about computing

George Boole's invention



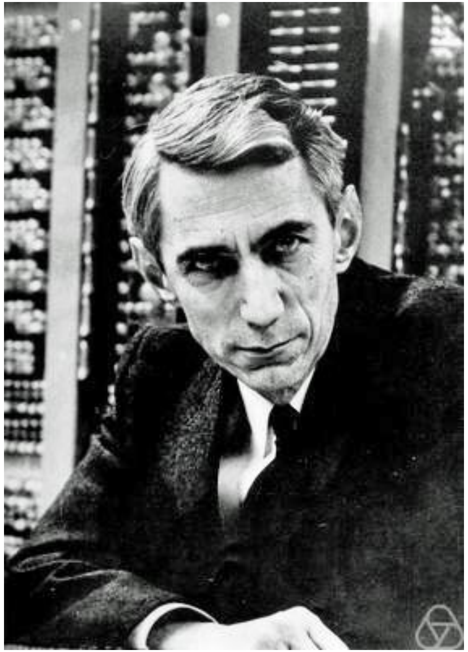
George Boole, 1815-1864.
The founder of Boolean Algebra

In 1854, George Boole introduced a system of logic operating on binary variables in his book - *An Investigation of the Laws of Thoughts*.

- It is a kind of "symbolic language"
- Reducing the logic into mathematical operations

Boole's work was extended by other mathematicians to form what is known as Boolean algebra.

From Boolean algebra to Electrical Circuits



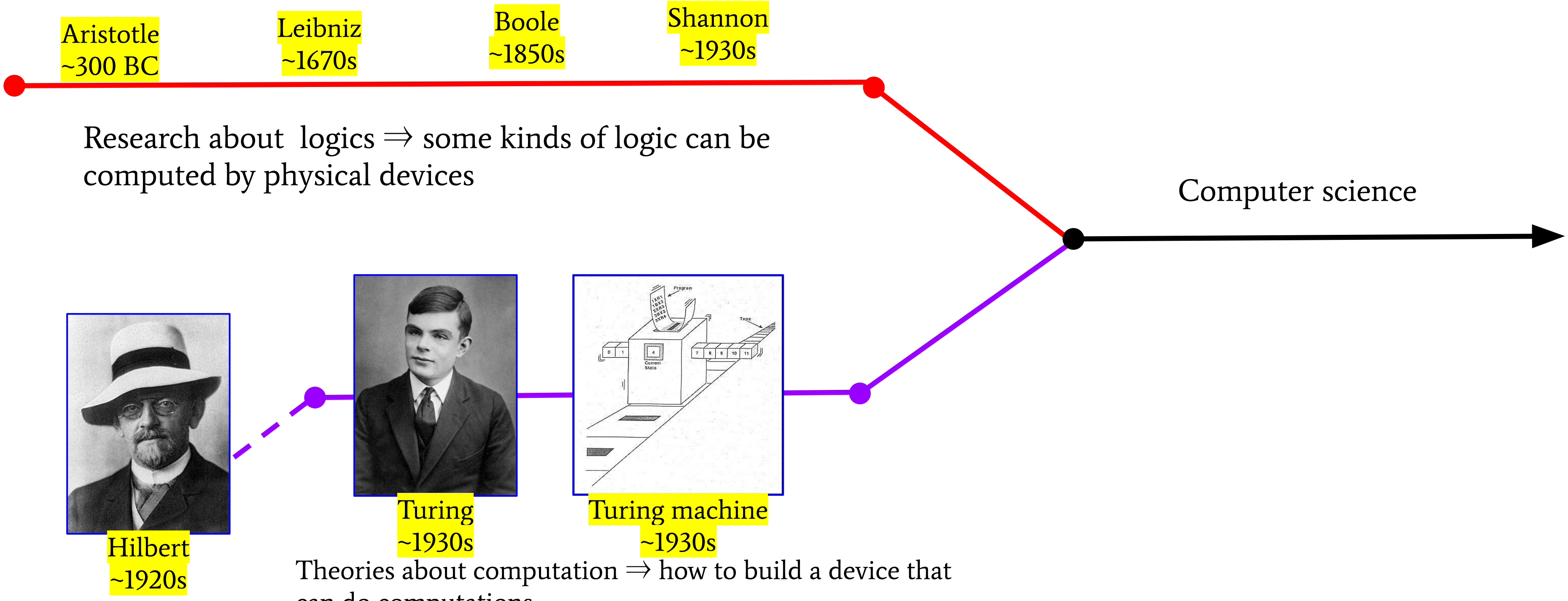
Claude Elwood Shannon, 1916-2001.
“The father of information theory”

In 1938, Shannon showed that it was possible to build electrical circuits that are equivalent to expressions in Boolean algebra.

- He showed that there is a one-to-one correspondence between the concepts of Boolean logic and some electrical circuits which are now called logic gates.
- He also defined “bit”, the measure of information.

Shannon’s work allows us to compute logic by circuits.

On the other hand ...



Aristotle
~300 BC

Leibniz
~1670s

Boole
~1850s

Shannon
~1930s

Research about logics ⇒ some kinds of logic can be computed by physical devices

Computer science

Hilbert
~1920s

Turing
~1930s

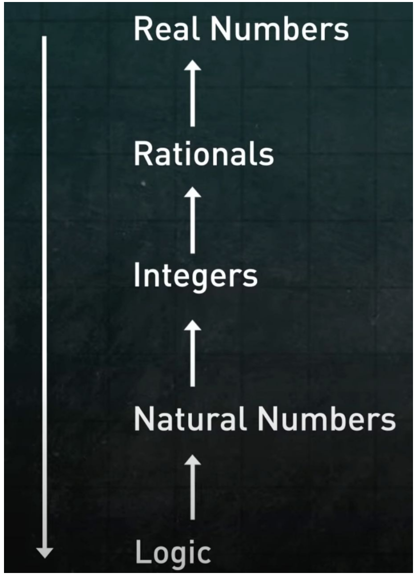
Turing machine
~1930s

Theories about computation ⇒ how to build a device that can do computations

Hilbert's program https://www.youtube.com/watch?v=KTUVdXI2vng

David Hilbert, 1862-1943

Hilbert believed that mathematics could be formulated on a solid and complete logical foundation.



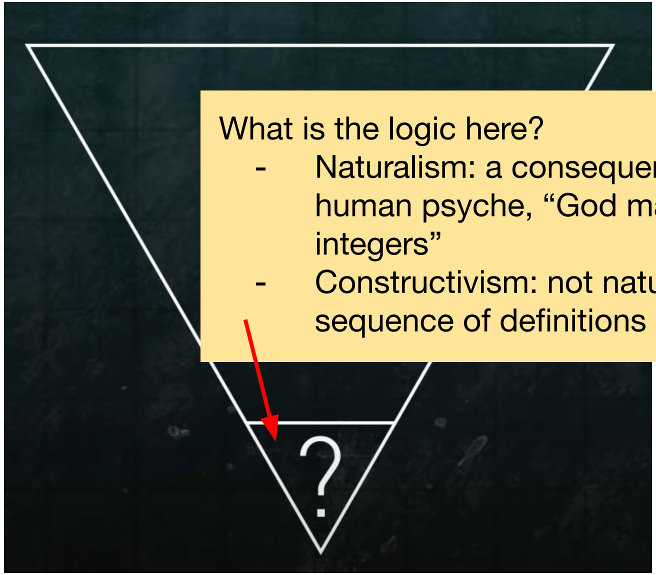
Real Numbers

Rationals

Integers

Natural Numbers

Logic



What is the logic here?

- Naturalism: a consequence of human psyche, "God made the integers"
- Constructivism: not natural, a sequence of definitions

Hilbert’s program

Hilbert plan to explain mathematics’ logic by proving the following propositions:

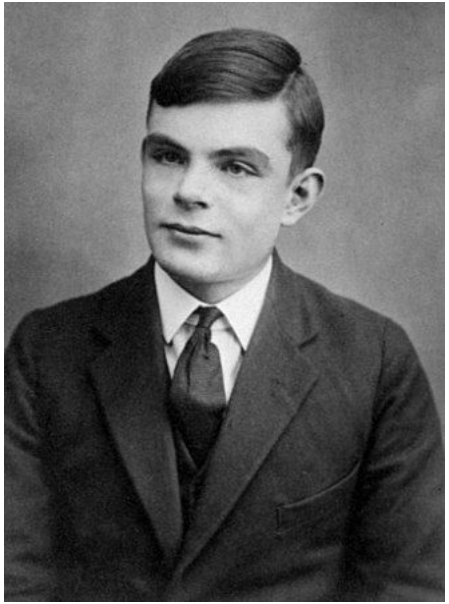
- The axioms should be consistent (Consistency)
- The axioms should be complete (Completeness)
- Entscheidungsproblem (Decidability, German for “decision problem”)

(Kurt Godel, an Austrian logician, proved the first two properties were false in 1931.)

Turing and his Turing machine

Turing, in 1937, showed the Entscheidungsproblem was false.

- To prove it, he designed an algorithm to do computation concretely, which is now called Turing machine.
- The machine can manipulate almost all known mathematical problems (he also pointed out its limitation⇒ there are some problems that are not computable).



Alan Turing, 1912 - 1954

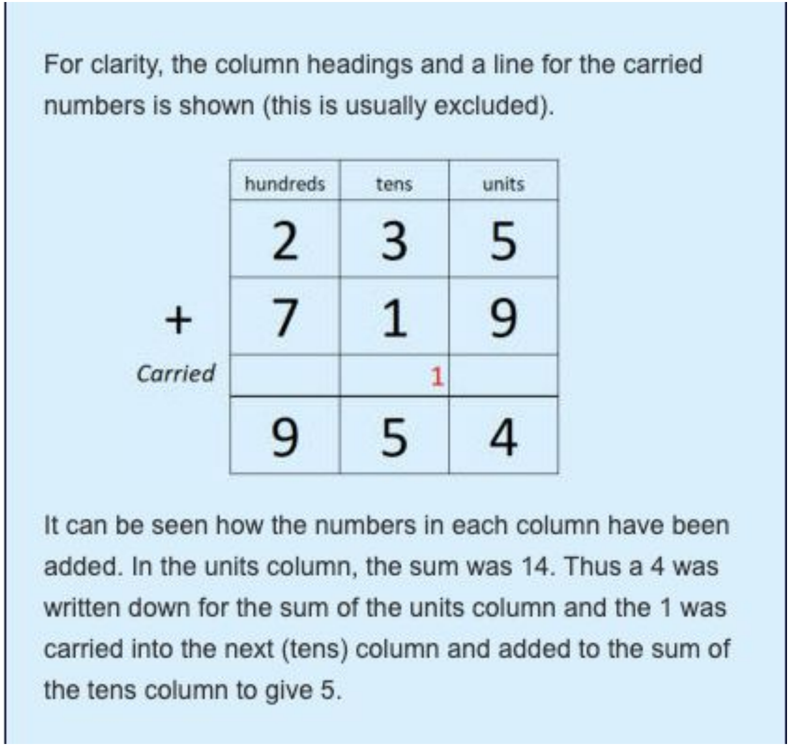
What is a Turing machine?

Firstly, let's think about how we do calculation.

- **Writing down** the numbers on paper
- **Handling** digits from the most right column to the most left column
- **Writing down** the output digit
- If needed, we **carry** some digit to the next column

E.g., adding: read 5 → read 9 → add them → put the result (4) to the cell at the left lower corner ……

- Replacing the numbers by some other symbols, we can do the computation (you defined) in the same way.

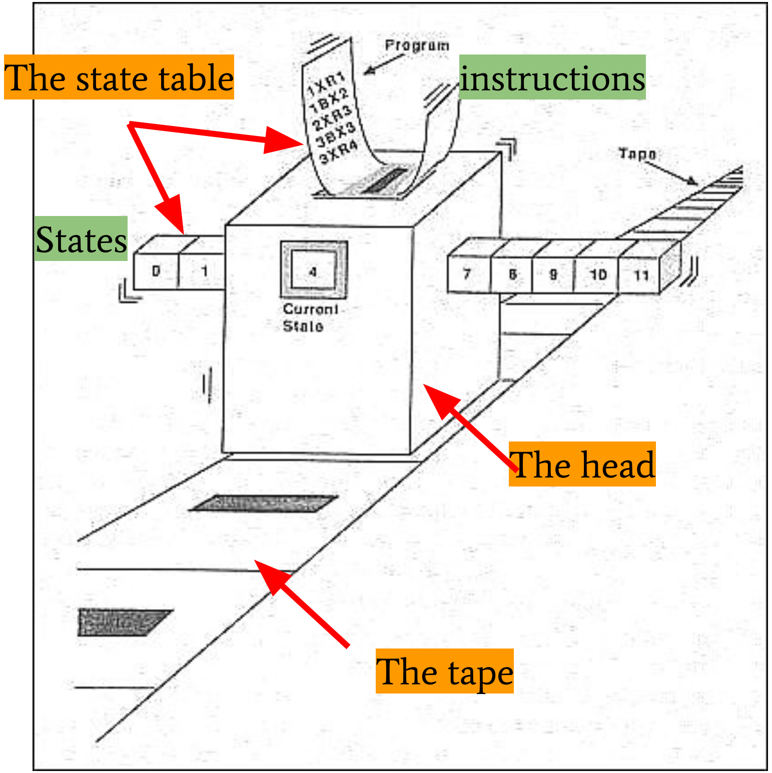


For clarity, the column headings and a line for the carried numbers is shown (this is usually excluded).

	hundreds	tens	units
	2	3	5
+	7	1	9
Carried		1	
	9	5	4

It can be seen how the numbers in each column have been added. In the units column, the sum was 14. Thus a 4 was written down for the sum of the units column and the 1 was carried into the next (tens) column and added to the sum of the tens column to give 5.

What is a Turing machine?



The state table

instructions

States

The head

The tape

[A Turing machine - Overview]

Turing machine is **a hypothetical** device that simulates human computation. It contains the following components:

- An infinitely-long tape (like the paper we use)
  - The tape is divided into squares; each can be written with symbols.

- A head over the tape (for “writing” and “carrying”)
  - Read the symbol on the square under it
  - Edit the symbol by writing a new one or erasing it
  - Move forward or back by one square on the tape

- A table which stores (like the “handling”)
  - States of the machine
  - Instructions (i.e., operations corresponding to the states.)

A workable Turing machine

https://www.youtube.com/watch?v=E3keLeMwfHY

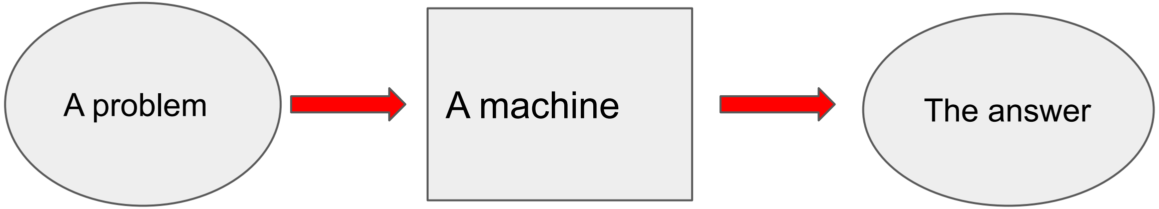
It only has three operations:

- Read the symbol on the square under the head
- Edit the symbol by writing symbol or erasing it
- Move the tape left or right by one square

Note: The machine has to be programmed to fulfil a task. Programmers should design the algorithm that can run on this machine, i.e., they can only use the above mentioned operations.

Turing machine

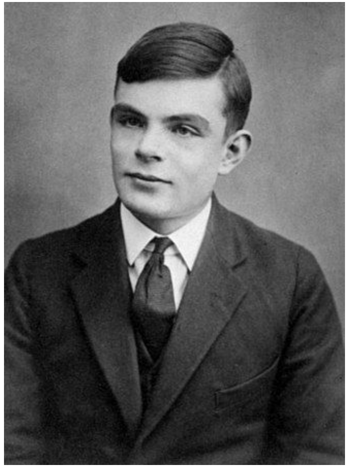
- It is an ideal model that maps an input problem to its answer.



A problem

A machine

The answer



Alan Turing, 1912 - 1954

- It is a “function” which can represent a solution with several simple operations.
- Programming (a Turing machine) is to describe the solution with these simple operations (provided by the machine).

Turing complete

- Turing complete: If a device can simulate a Turing machine, we say it is Turing complete.

- A Turing machine is **not** necessary to be a real physical device
    - A programming language can be a Turing machine, e.g., Python.
        - A Python list can be considered as the tape (though its length depends on the size of memory of the computer)
        - Each element in the list can be read or written.

Note: we sometimes call abstract machine as automata.

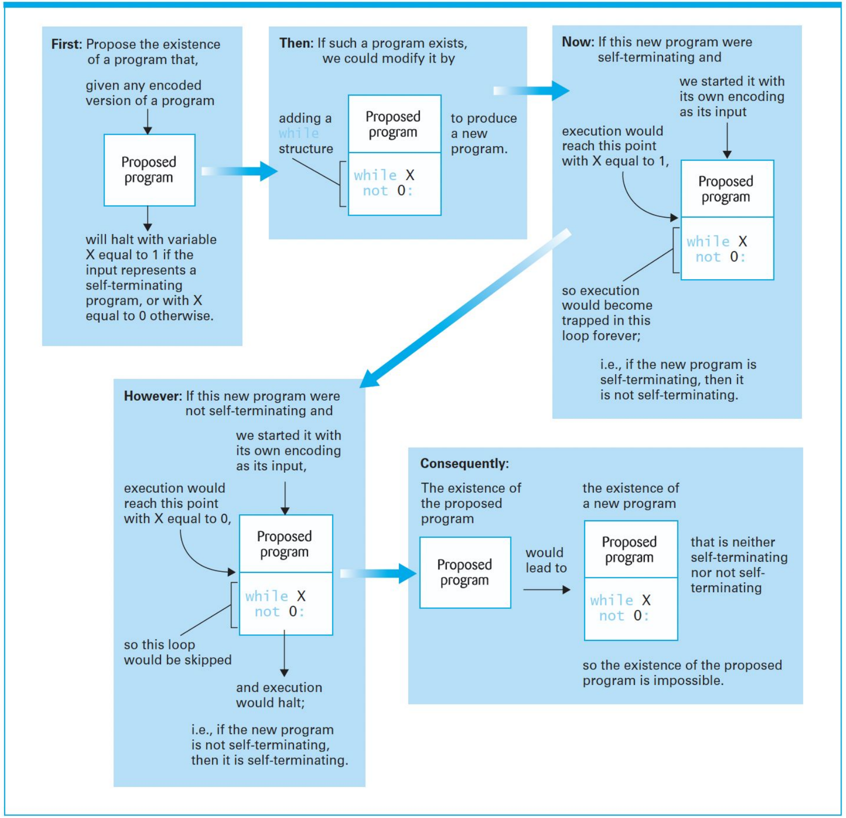
The limitation of Turing machine

There are problems that a Turing machine cannot give an output.

- Halting problem: to predict in advance whether a program will stop if started under certain conditions. (unsolvable, proved by contradiction)

In other words, there are problems, e.g, the halting problem, though they can be programmed, they lie beyond the capabilities of computers.
https://brilliant.org/wiki/halting-problem/

Figure 12.7 Proving the unsolvability of the halting program



First: Propose the existence of a program that,

given any encoded version of a program

Proposed program

will halt with variable X equal to 1 if the input represents a self-terminating program, or with X equal to 0 otherwise.

Then: If such a program exists, we could modify it by

adding a while structure

Proposed program

```
while X
 not 0:
```

to produce a new program.

Now: If this new program were self-terminating and

we started it with its own encoding as its input

execution would reach this point with X equal to 1,

Proposed program

```
while X
 not 0:
```

so execution would become trapped in this loop forever;

i.e., if the new program is self-terminating, then it is not self-terminating.

However: If this new program were not self-terminating and

we started it with its own encoding as its input,

execution would reach this point with X equal to 0,

Proposed program

```
while X
 not 0:
```

so this loop would be skipped

and execution would halt;

i.e., if the new program is not self-terminating, then it is self-terminating.

Consequently:

The existence of the proposed program

Proposed program

would lead to

the existence of a new program

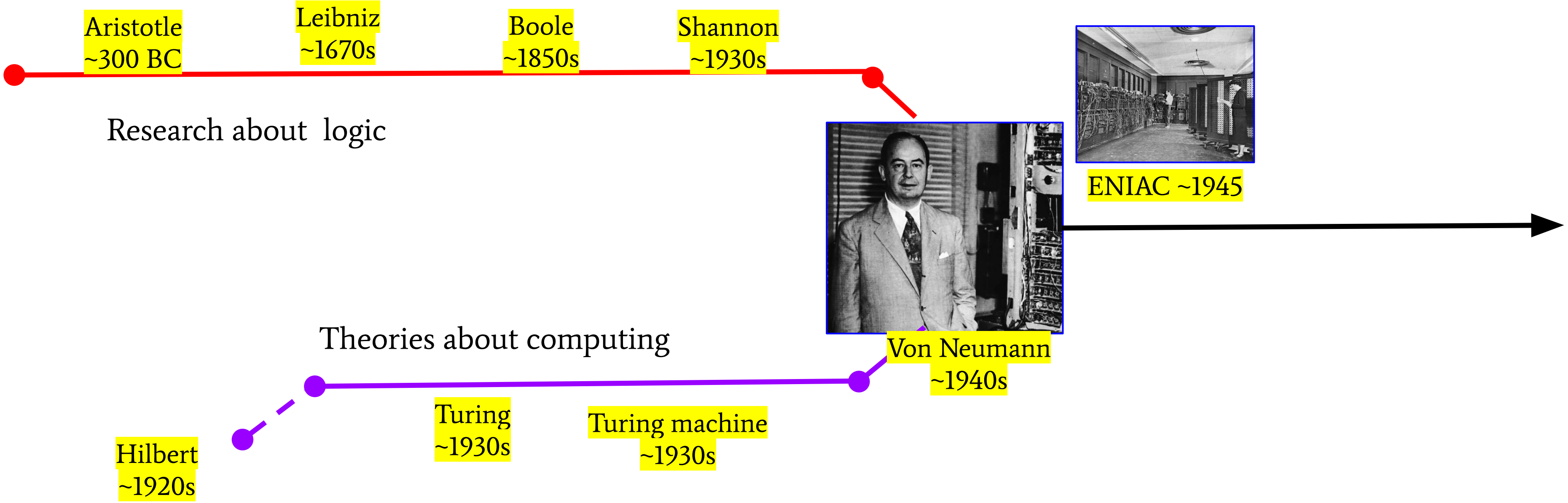
Proposed program

```
while X
 not 0:
```

that is neither self-terminating nor not self-terminating

so the existence of the proposed program is impossible.

The birth of computer



Aristotle ~300 BC

Leibniz ~1670s

Boole ~1850s

Shannon ~1930s

Research about logic

ENIAC ~1945

Von Neumann ~1940s

Theories about computing

Hilbert ~1920s

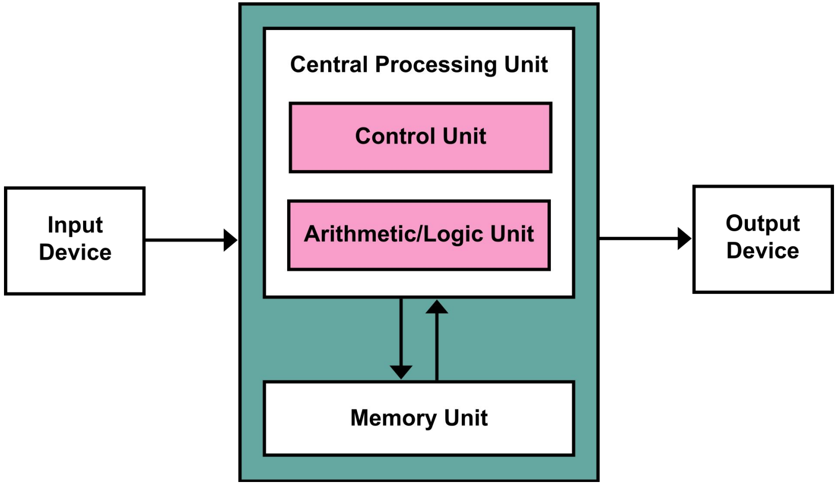
Turing ~1930s

Turing machine ~1930s

Modern computer: Von Neumann Architecture

Von Neumann architecture: a Turing machine made by electric circuits.

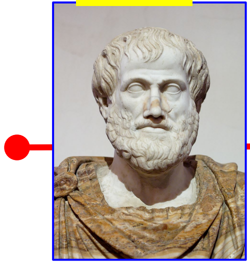
- Tape → memory
- Head → CPU
- State table → instructions and programs



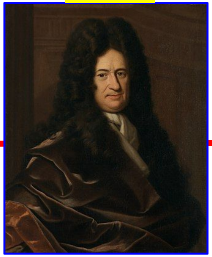
Von Neumann architecture

A history of computer science

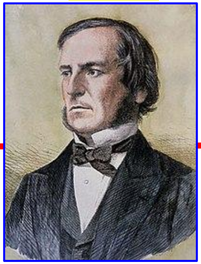
Aristotle
~300 BC



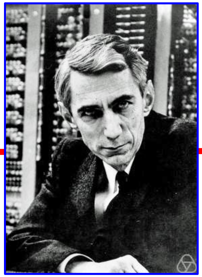
Leibniz
~1670s



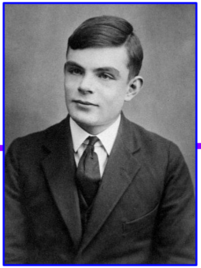
Boole
~1850s



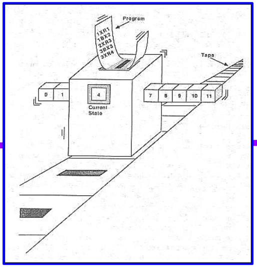
Shannon
~1930s



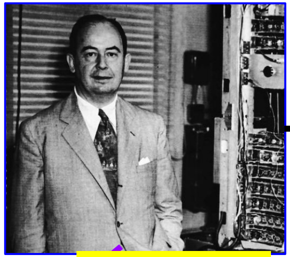
Hilbert
~1920s



Turing
~1930s

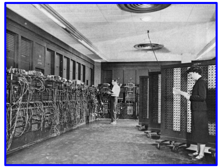


Turing machine
~1930s



Von Neumann
~1940s

Computer science



ENIAC ~1945

Supercomputer

Mac and PC
~1980s

?

Agenda

- The history of computer science
  - The origins
  - Turing machine
- **Boolean algebra**
  - **Variables, operators, and truth table**
  - **Numbers in boolean algebra**
- Appendix
  - Notations for integers and fractions

Boolean variables

- A Boolean variable can only have two values: True and False
  - We usually use 1 to represent True, and 0 to represent False.

For example,

- p = “ Today is sunny.” is a Boolean variable because it can be True or False.
- q = “What time is it now?” is not a Boolean variable.

Three basic Boolean operators

- NOT (denoted by "¬")
  - The negation of a statement (denoted by ¬ p, also denoted by $\overline{p}$).
  - NOT "New York is the capital of USA" → "New York is not the capital of USA."

- AND (denoted by "×" or "$\wedge$")
  - If p is True and q is True, then p AND q is True; otherwise p AND q is False.
  - "New York is the capital of USA and Beijing is the capital of CHN."

- OR (denoted by "+" or "$\vee$")
  - If p is True or q is True, then p OR q is True. (i.e., p OR q is False when both p and q are False and is true otherwise.)
  - "New York is the capital of USA or Beijing is the capital of CHN."

Truth table

All possible values of a logical expression can be listed in a table.

Truth table of p AND q

p	q	$p \times q$
T	T	T
T	F	F
F	T	F
F	F	F

- each column shows the possible values of a logic variable/expression.
- each row shows a **possible** **combination** of truth values of logical variables and the result of the logic expression.

More examples of truth table

- A truth table should show **all** possible combinations.
- Each row in the truth table shows a **possible combination** of truth values of variables.

Truth table of *p* OR *q*

p	q	$p + q$
T	T	T
T	F	T
F	T	T
F	F	F

- A truth table contains 2 variables has 4 rows.

- How many rows does a truth table of *n* variables have?

2^n

Truth table

NOT *p*

p	$\neg p$
T	F
F	T

Truth table

$p \oplus q$, called exclusive OR (XOR, “$\oplus$”, called “circle plus”), is defined as $p \oplus q = (p + q) \times (\neg p + \neg q)$.

p	q	$(p + q)$	$(\neg p + \neg q)$	$p \oplus q$
T	T	T	F	F
T	F	T	T	T
F	T	T	T	T
F	F	F	T	F

- $p \oplus q$ is true when exactly one of p and q is true.

Boolean Algebra: Logical equivalences

Let *p*, *q*, and *r* be three boolean variables.

- Commutative laws:
  - $p \times q = q \times p$
  - $p + q = q + p$
- Associative laws:
  - $(p \times q) \times r = p \times (q \times r)$
  - $(p + q) + r = q + (p + r)$
- Distributive laws:
  - $p \times (q + r) = (p \times q) + (p \times r)$
  - $p + (q \times r) = (p + q) \times (p + r)$

We can prove them by using truth table.

- If two Boolean expressions have the same truth table, then they are equivalent.

De Morgan’s Laws

Augustus De Morgan,1806-1871, was a British mathematician and logician.

Two Boolean algebraic equations are called De Morgan’s Laws.

- $\neg (p \times q) = \neg p + \neg q$
- $\neg (p + q) = \neg p \times \neg q$

Truth tables of De Morgan’s laws

The logical equivalences are proved by using truth tables.

p	q	$p+q$	$\neg(p + q)$	$\neg p$	$\neg q$	$\neg p \times \neg q$	$\neg p + \neg q$	$p \times q$	$\neg(p \times q)$
T	T	T	F	F	F	F	F	T	F
T	F	T	F	F	T	F	T	F	T
F	T	T	F	T	F	F	T	F	T
F	F	F	T	T	T	T	T	F	T

Simplify logic expressions with De Morgan’s laws

We can use De Morgan’s laws to simplify logical expressions.

For example,

$$\neg\,(p+(\neg\,p\times q\,)\,)=\neg\,p\times\neg\,(\neg\,p\times q\,) \qquad \text{\# De Morgan's laws}$$

$$=\neg\,p\times[\neg\,(\neg p)+\neg\,q\,] \quad \text{\# De Morgan's laws}$$

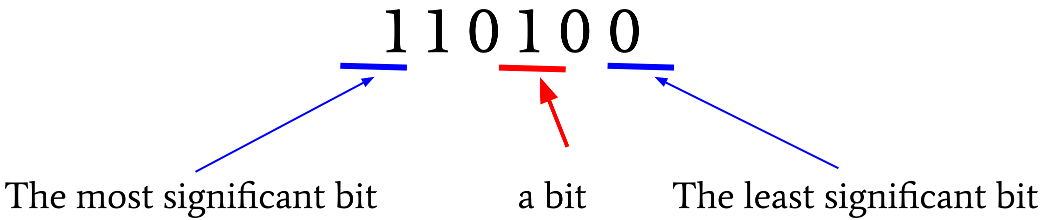
$$=\neg\,p\times(p+\neg\,q)$$

$$=\neg\,p\times p+\neg\,p\times\neg\,q \quad \text{\# distributive law}$$

$$=\neg\,p\times\neg\,q$$

Binary strings

A binary string (also called binary number) is a set of columns of 0s and 1s:



1 1 0 1 0 0

The most significant bit

a bit

The least significant bit

- A **bit** means a binary digit, representing one of two values: 0 or 1.

The state (on/off) of a switch is a bit.

Describing decimals in Boolean algebra

We can represent everything we have in math by using True and False.

Let's take integers as an example.

S1	0	0
S2	0	1
S3	1	0
S4	1	1

Mapping the states to decimals, we have

Base 2	Base 10
00	0
01	1
10	2
11	3

If we want to represent 7_{10} in binary code, how many "switches" do we need?

*Actually, there are several ways to represent numbers as binary strings, please see the appendix.

Converting binary to decimal

Base 2 to base 10: $10110_2 = ?_{10}$

- In a binary number, each column has twice the weight of the previous column, so for i-th column, adding $D_i \times 2^{i-1}$.

$$10110_2 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1\times 2^1 + 0 \times 2^0 = 22_{10}$$

What is the decimal value of 1011001_2?

What is the decimal value of 10110010111_2?

Tedious to work it out on paper?

- Let's write a piece of code to convert the input binary into decimal.
- Note: We assume the input binary number is a Python string.
- Or, using the Python built-in functions `int/bin` for the conversion

Converting decimal to binary.

Base 10 to base 2: $84_{10} = ?_2$

- Repeatedly divide the number by 2; the remainder goes in each column (from right to left).
  - 84/2 = 42, so 0 goes in the most right column
  - 42/2 = 21, so 0 goes in the 2nd right column
  - 21/2 = 10, with a remainder of 1 going in the 3rd right column
  - 10/2 = 5, so, 0 goes in the 4th right column
  - 5/2 = 2, with a remainder of 1 going in the 5th right column
  - 2/2 = 1, so, 0 goes in the 6th right column
  - 1/2 = 0, with a remainder of 1 going in the 7th right column

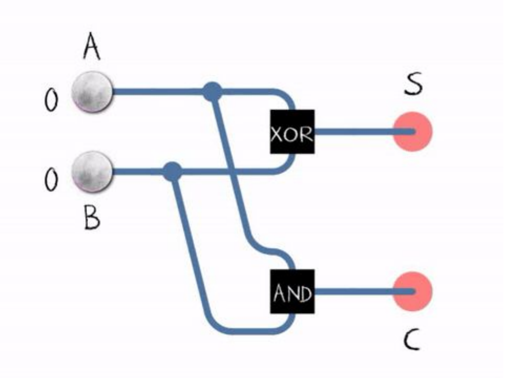
  So, $84_{10} = 1010100_2$.

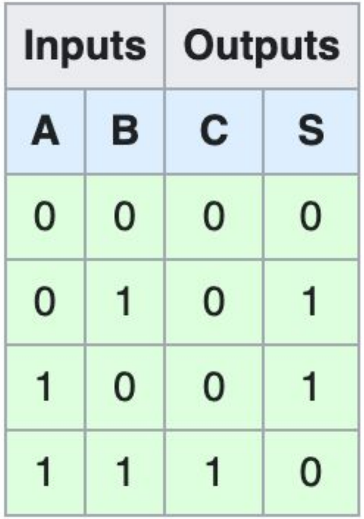
Adding numbers in Boolean algebra

Variable	Decimal	Binary
A	2	10
B	3	11
SUM	5	101

10
+ 1 11
101

- We add A and B **digit by digit** from right to left;
- Adding two digits has two outputs: S (the sum), and C (the carry);
- We can list all possible inputs and outputs as a truth table.
- S = A **XOR** B
- C = A **AND** B

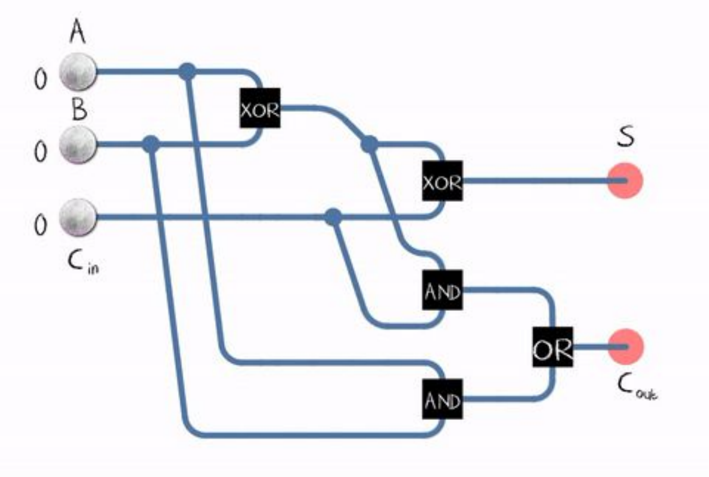


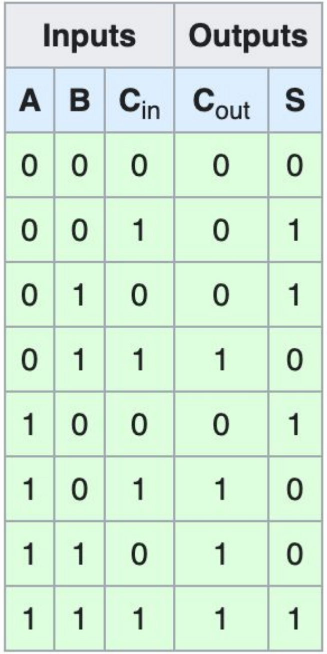


Inputs		Outputs	
A	**B**	**C**	**S**
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Adding numbers in Boolean algebra

- From the second two digits we are going to add, we need to add three digits: one from A, one from B, and the carry C we get previously; and, we can have truth table for it.
- For the S, it is obtained by (A XOR B) XOR C_{in};
- For the C_{out}, it is obtained by (A AND B) OR [(A XOR B) AND C_{in}];





Inputs			Outputs	
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Appendix:

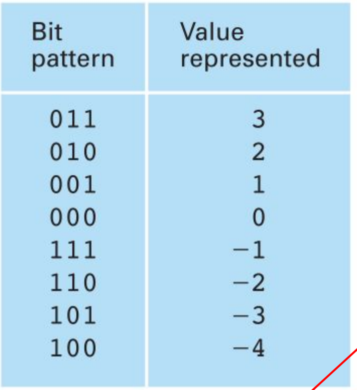
- Notations for integers and fractions

Two’s complement notation

The most popular system for representing integers in today’s computers

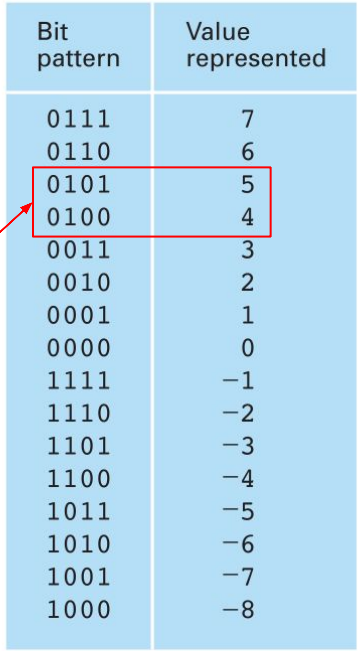
- Positive values are obtained by starting with a string of 0s of a preset length and counting in binary until the pattern consisting of a single 0 followed by 1s
- Negative values are obtained by starting with a string of 1s of the preset length and counting backward in binary until the pattern consisting of a single 1 followed by 0s.

a. Using patterns of length three



Bit pattern	Value represented
011	3
010	2
001	1
000	0
111	−1
110	−2
101	−3
100	−4

b. Using patterns of length four



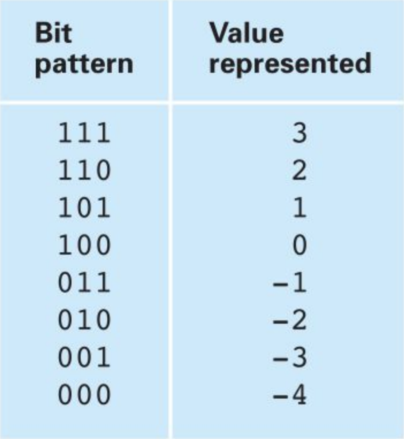
Bit pattern	Value represented
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	−1
1110	−2
1101	−3
1100	−4
1011	−5
1010	−6
1001	−7
1000	−8

Overflow: 5 + 4 = -7
(because 9 is out of [7, -8])

Excess notation

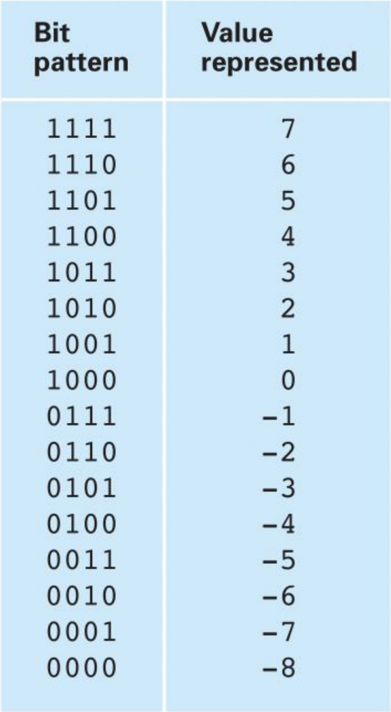
- Based on the given length, write down all the different bit patterns in order
- Find the pattern with only a 1 and that 1 is its most significant bit. Let it to represent 0.
- The patterns following this are used to represent 1, 2, 3, …
- The patterns preceding it are used to represent -1, -2, -3, …

Using pattern of length three



Bit pattern	Value represented
111	3
110	2
101	1
100	0
011	−1
010	−2
001	−3
000	−4

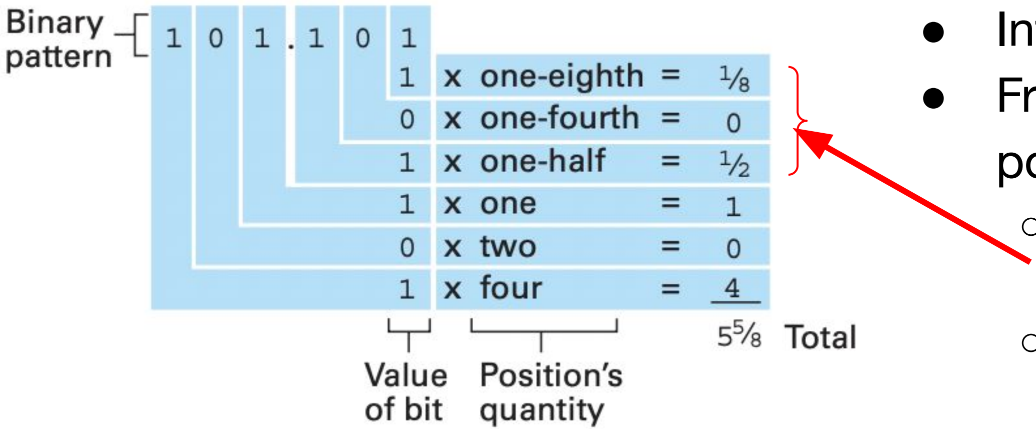
Using pattern of length four



Bit pattern	Value represented
1111	7
1110	6
1101	5
1100	4
1011	3
1010	2
1001	1
1000	0
0111	−1
0110	−2
0101	−3
0100	−4
0011	−5
0010	−6
0001	−7
0000	−8

Fractions in Binary

Decoding the binary representation 101.101



1 x one-eighth = ⅛

0 x one-fourth = 0

1 x one-half = ½

1 x one = 1

0 x two = 0

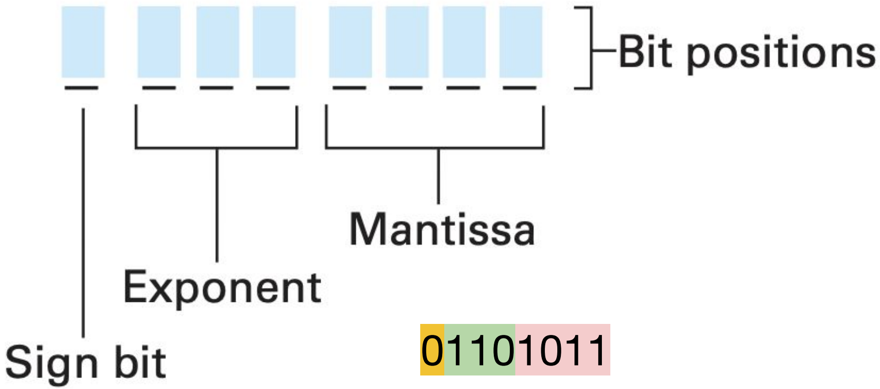
1 x four = 4

5⅝ Total

- Using a **radix point** as the decimal point in decimal notation
- Integer part: the left of the point
- Fractional part: the right of the point
  - Each position is assigned a fractional quantity.
  - The first position is divided by 2, and the seconde is divided by 4, and so on.

Floating-point notation

A format for representing negatives and fractions.



- Sign bit: 0 means the value is nonnegative and 1 means it is negative.
- Exponent field: tells how to move the radix, interpreted as an integer in excess notation. If positive, move right; if negative, move left.
- Mantissa field: tells the digits of the value, starts with a radix

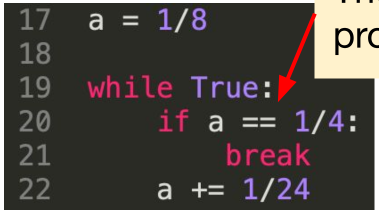
01101011

Sign bit: 0 → positive
Exponent: 110 → 2 → moving radix right by 2 digits
Mantissa: 1011→ .1011 (radix is initialized at the most left hand side)
So, we have 10.11 → 2¾

Round-off error

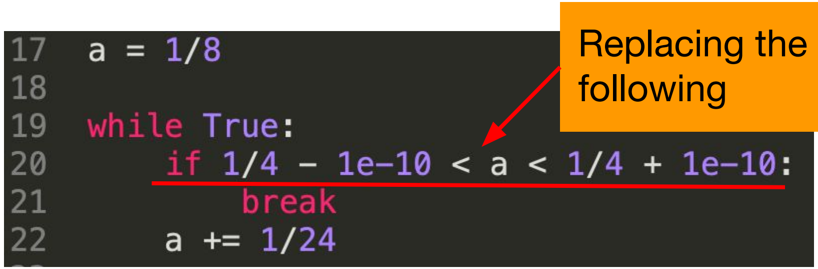
Floating-point notation is an approximation of the numbers. Due to the size limits of the mantissa, not all numbers can be represented precisely.

- ⅛ + 1/24 +1/24 + 1/24 = ¼
- When you need to check if two float-point numbers are equal to each other, you'd better use b - ε < a < b + ε.



```python
a = 1/8

while True:
    if a == 1/4:
        break
    a += 1/24
```

The code has a problem!



```python
a = 1/8

while True:
    if 1/4 - 1e-10 < a < 1/4 + 1e-10:
        break
    a += 1/24
```

Replacing the "==" by the following

Supplementary

Watching BBC: history of computers

- It takes about many anecdotes in the development of computers

- More about floating-number notation & the common 32-/64-bit systems
  - Section 4.1/4.3 in this tutorial
    https://www3.ntu.edu.sg/home/ehchua/programming/java/datarepresentation.html#zz-4.1