From Zero To One

An introduction to the Boolean algebra and number systems

About us

Instructor: Li Guo lg154@nyu.edu
Grader: Yunjie(Chloe) Song yunjie.song@nyu.edu
Learning Assistant: Yifan Zhuo yz4044@nyu.edu
Houze Liu hl2979@nyu.edu

Office hours:
Li Guo: Mon 3:15-4:15PM and Wed 11:15AM-12:15PM at Room 1157
Yifan Zhuo: Thu 2:00PM-3:00PM
Houze Liu : Fri 11:00AM-12:00PM

Why study computer science?

Course structure

Seg 1: computer architecture with a history		
Computer architecture/history	Basic Python	Quiz 1
Seg2: algorithmic thinking and practice		
Algorithm/complexity families	Python: OOP	Midterm
Seg 3: data science intro		
Stats/prob/machine learning	Chat system	Quiz 2
Seg 4: advanced topics		
Deep learning		Final

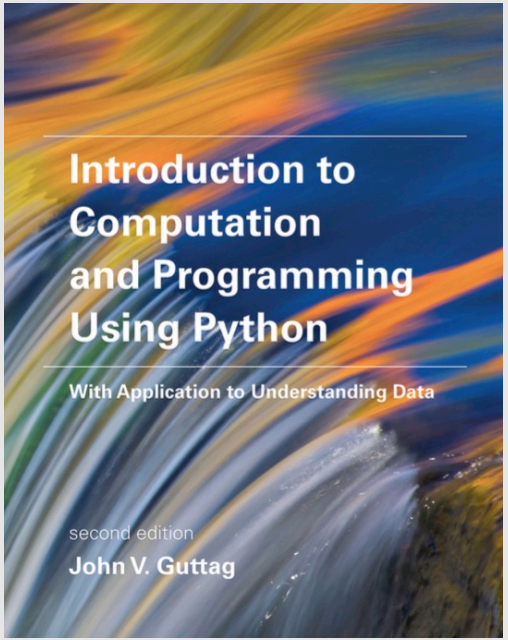
Unit project

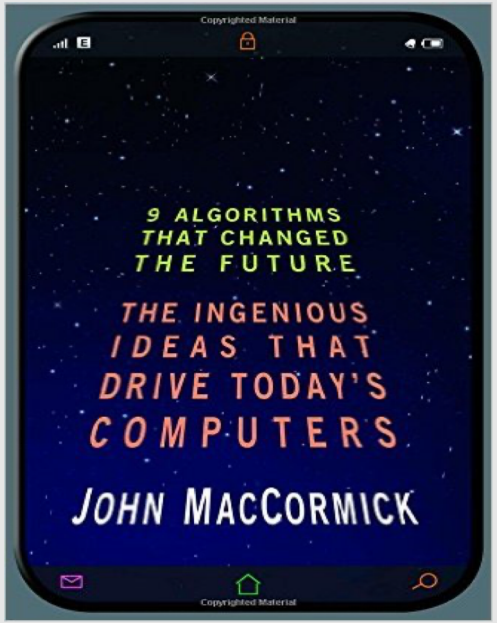
A complete distributed chat system

Evaluation

Quizzes	15%
Midterm	15%
Final	20%
Unit projects	15%
Homework	15%
Presentation projects	20%
Bonus	10%

Textbooks


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


GLOBAL EDITION
Starting Out with Python
THIRD EDITION
Tony Gaddis
ALWAYS LEARNING
PEARSON

Rules

- Laptops (mandatory for labs)
- Tablets
- Questions, comments and spot mistakes
- Attend class on time!

- Discussion outside the topics of the class
- Enter after **15 minutes** of class start

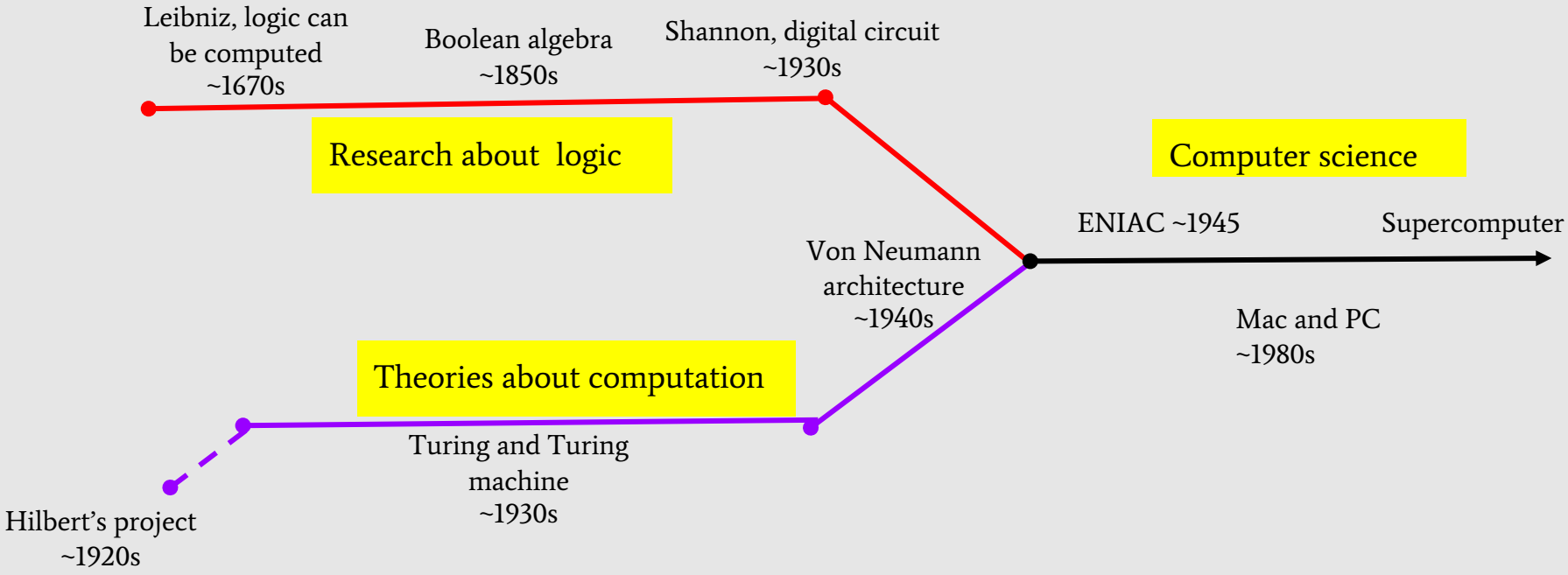
Q&A

Agenda

- The history of computer science
  - The origins
  - Turing machine
- Boolean algebra
  - Variables, operators, and truth table
  - De Morgan’s Laws (optional)
- Numbers systems
  - Binary numbers
  - Converting between number systems

history of computers

A history of computer science

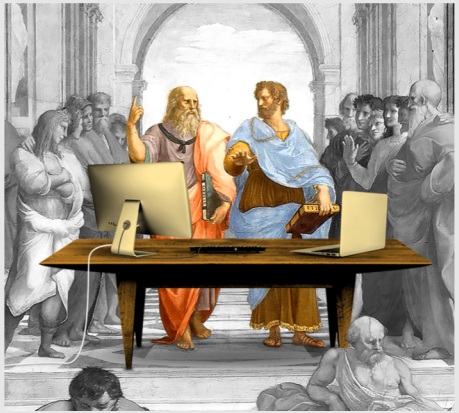


Leibniz's idea: logic can be computed

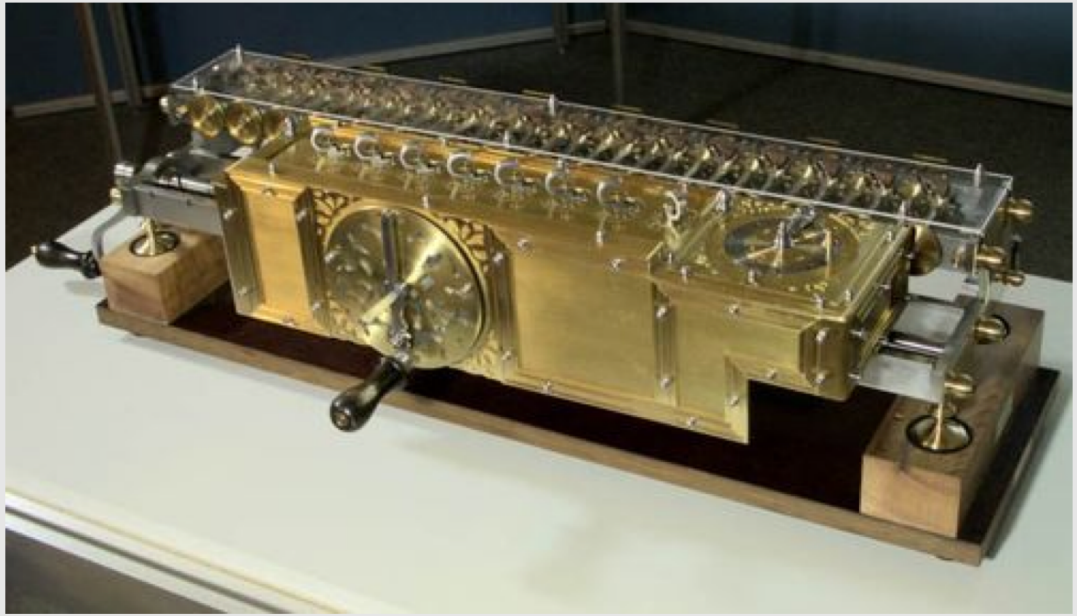
Gottfried Wilhelm von Leibniz,
1646 - 1716

"If controversies were to arise, there would be no more need of 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 witness if they like): Let us calculate."

--- Leibniz, in Russell's translation



Leibniz's Stepped Reckoner

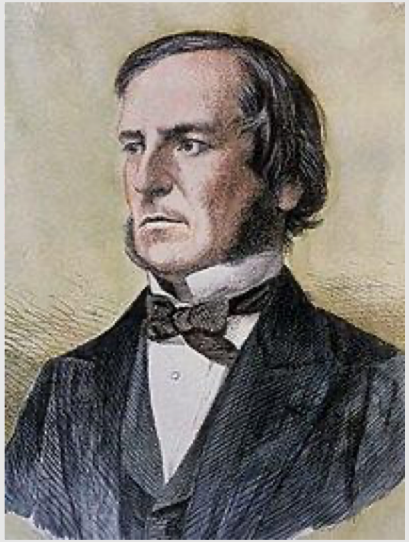


Stepped Reckoner (Wikipedia)

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

It took him more than 20 years to build it.

George Boole's invention



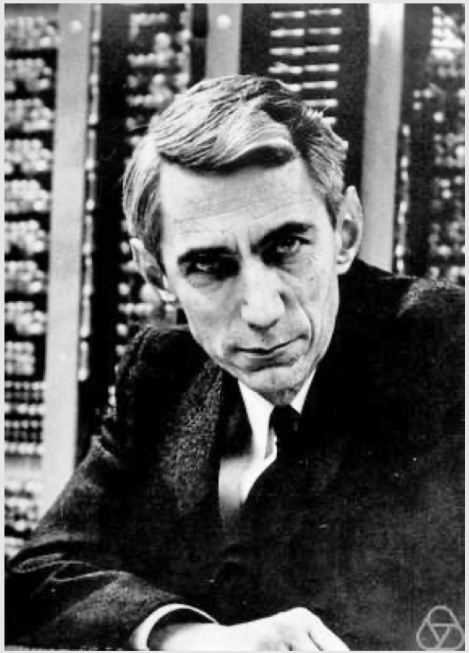
George Boole, 1815-1864.
The founder of Boolean Algebra

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 equivalent to expression 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.

Hilbert's project

David Hilbert, 1862-1943

Is there a systematic way to compute the solution to any mathematical problem

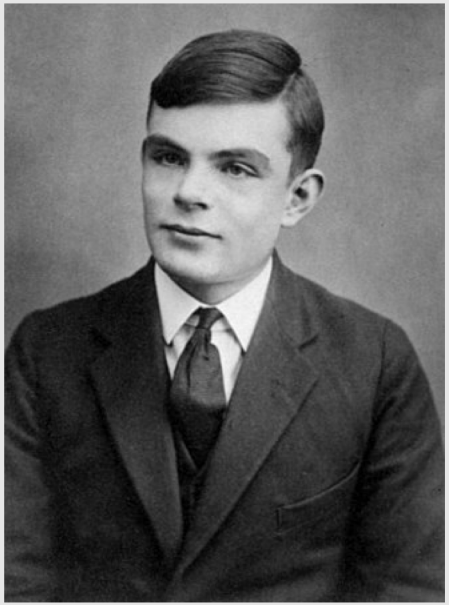
Entscheidungsproblem (German for "decision problem")

Turing and his Turing machine

Turing, in 1937, proposed a method that shown the Entscheidungsproblem was false.

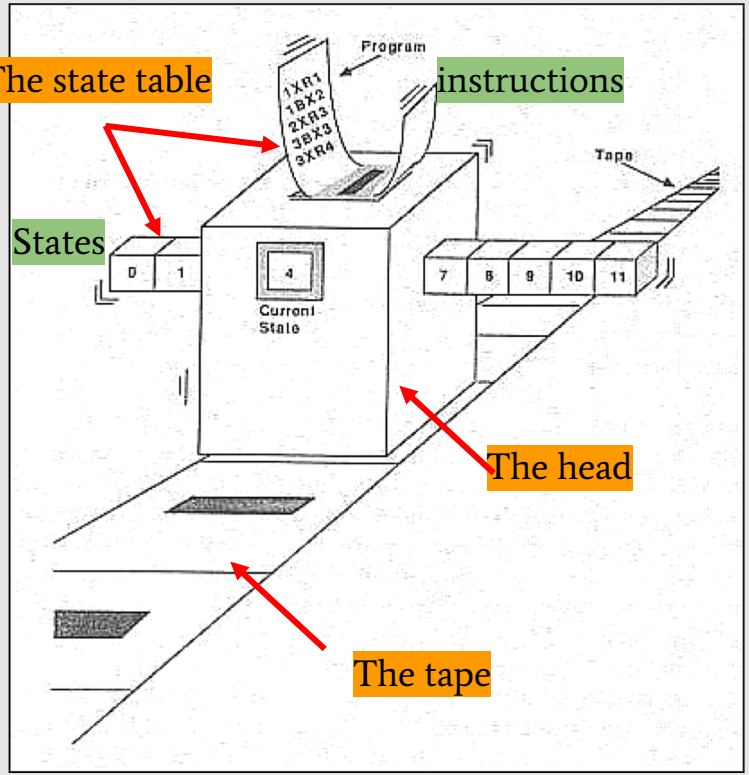
- He designed a model to describe what computation is, which is now called Turing machine.

- The machine can manipulate almost all known mathematical problems (he also pointed out its limitation).



Alan Turing, 1912 - 1954

What is a Turing machine?



A Turing machine - Overview

Turing machine is a model, a hypothetical device that can do computation.

It contains the following components:

- An infinitely-long tape
  - The tape is divided into squares; each can be written with symbols.
- A head over the tape:
  - 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
  - States of the machine
  - Instructions (i.e., operations corresponding to the states.)

Halting problem: the limitation of Turing machine

The Halting problem – Given a program/algorithm will it ever halt or not?

- **Not** all mathematical problems can be solved via computation (i.e., the Turing machine).

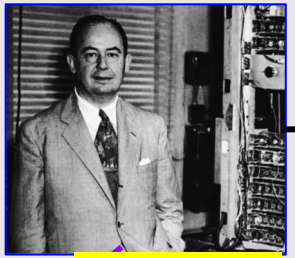
The birth of computer

Leibniz
~1670s

Boole
~1850s

Shannon
~1930s

Research about logic



Von Neumann
~1940s

ENIAC ~1945

Theories about computing

Hilbert
~1920s

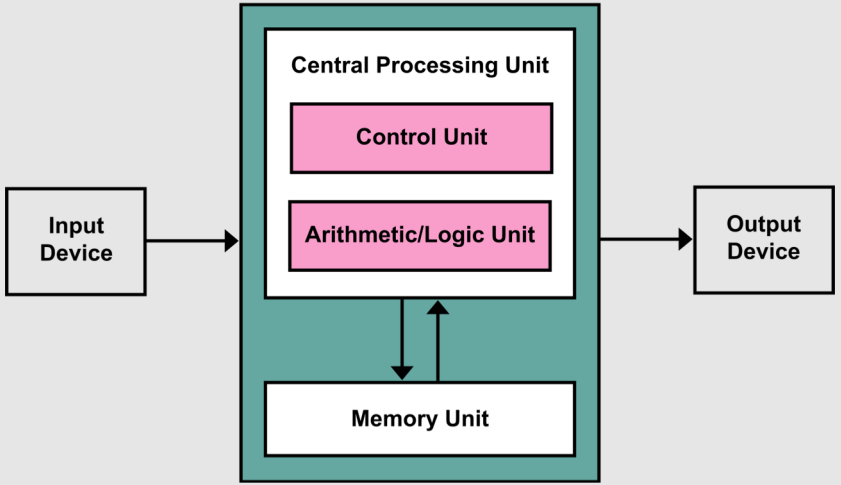
Turing
~1930s

Turing machine
~1930s

Modern computer: Von Neumann Architecture

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

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

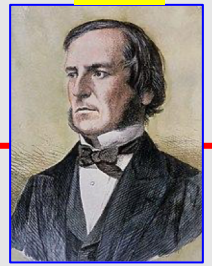


Von Neumann architecture

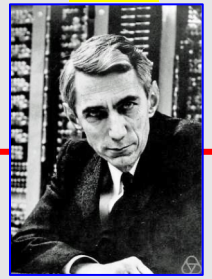
A history of computer science

Leibniz
~1670s

Boole
~1850s



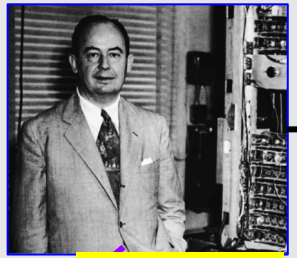
Shannon
~1930s



Computer science

ENIAC ~1945

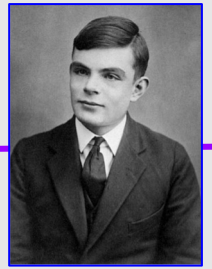
Supercomputer



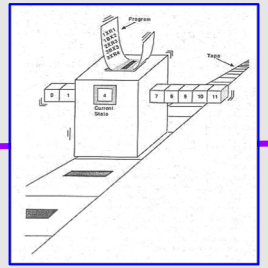
Von Neumann
~1940s

Mac and PC
~1980s

Hilbert
~1920s



Turing
~1930s



Turing machine
~1930s

Boolean algebra

Boolean variables

- The Boolean variables (i.e., operands) 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 $\neg p$, 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 "∧")
  - 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." (False)

- OR (denoted by "+" or "∨")
  - 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." (True)

Truth table

A truth table is a list of **all** possible values of a logical expression.

Truth table of p AND q

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

Truth table of p OR q

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

Truth tables

NOT p

p	$\neg p$
T	F
F	T

Truth table

p ⊕ q, called exclusive OR (XOR, "⊕"), is defined as p ⊕ q = $(p + q) \times (\neg p + \neg q)$.

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

- p ⊕ 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.

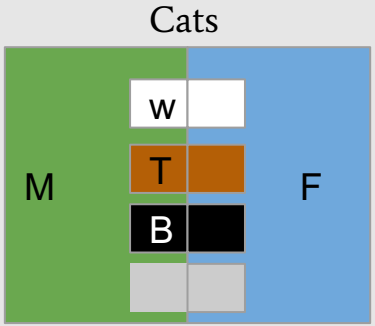
- If two Boolean expressions have the same truth table, then they are equivalent.

Deduction and Reasoning

John wanted a cat. One day, he went into a pet shop and said to the assistant, “ I would like to buy a male cat whose color is white or tan; or, a female cat whose color is anything except white; or a black cat of any gender.” The assistant said, “ We have a gray male cat.”. Do you think John would buy it or not?

- M: cats are male
- F: cats are female
- W: cats’ color is white
- T: cats’ color is tan
- B: cats’ color is black

The cat John required:
$M \times (W + T) + F \times (\neg W) + B$



The cat available:
M = 1,
F = 0,
W = 0,
T = 0,
B = 0

Plug into the expression of conditions, we have
$1 \times (0 + 0) + 0 \times (\neg 0) + 0 = 0$.
Therefore, John wouldn’t buy that cat.

De Morgan’s Laws (optional)

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

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

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 expression with De Morgan's laws

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

For example,

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

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

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

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

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

Number systems

Why using binary number?

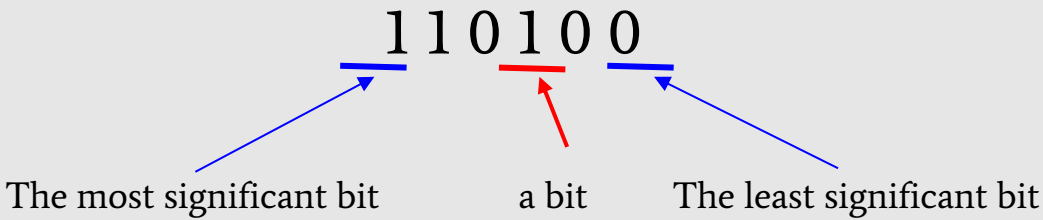
We are accustomed to work with decimal numbers (i.e., 0, 1, ..., 9), but in a digital system, it often works with binary numbers (i.e, 0, 1).

Why?

- Because a circuit only has 2 states (on/off).

Binary Numbers

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



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

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

Binary Code (cont.)

- Base 10:
  - Each digit has 10 states
- Base 2:
  - Each digit has 2 states

Base 10	Base 2
0	0
1	1
2	10
3	11
4	100
5	101
8	1000

What is 6 and 7 in binary code?

From Base2 to Base10

- Count the position from rightmost to left
  - For the i-th position, add $D_i \times 2^{i-1}$
  - E.g.: $101_b = 2^2 \times 1 + 2^1 \times 0 + 2^0 \times 1$

From Base2 to Base10

- Count the position from rightmost to left
  - For the i-th position, add $D_i \times 2^{i-1}$
  - E.g.: $101_b = 2^2 \times 1 + 2^1 \times 0 + 2^0 \times 1$

- Q: what's the value of 1011001 in binary?

- Too tedious to work it out on paper?
  - Try a **lazy** way: write a piece of code!

From Base2 to Base10

```
 8 bcode = '100001011010'
 9
10 def binary_to_int(binary_code):
11     result = 0
12     code_len = len(binary_code)
13     r_bcode = binary_code[::-1]
14     for i in range(code_len):
15         bit = int(r_bcode[i])
16         result += bit * 2**i
17         print('%d-th bit is %d: result = %d' % (i, bit, result))
18     return result
19
20 int_result = binary_to_int(bcode)
21 print(bcode, 'converts to', int_result)
```

```
In [9]: run b2int.py
0-th bit is 0: result = 0
1-th bit is 1: result = 2
2-th bit is 0: result = 2
3-th bit is 1: result = 10
4-th bit is 1: result = 26
5-th bit is 0: result = 26
6-th bit is 1: result = 90
7-th bit is 0: result = 90
8-th bit is 0: result = 90
9-th bit is 0: result = 90
10-th bit is 0: result = 90
11-th bit is 1: result = 2138
('100001011010', 'converts to', 2138)

In [10]: binary_to_int('10110010')
0-th bit is 0: result = 0
1-th bit is 1: result = 2
2-th bit is 0: result = 2
3-th bit is 0: result = 2
4-th bit is 1: result = 18
5-th bit is 1: result = 50
6-th bit is 0: result = 50
7-th bit is 1: result = 178
Out[10]: 178
```

The basic unit is not a bit! But a byte

1 0 1 1 0 0 1 0

- A 8 bit-long binary number, 10110010, can be represented as a row of 8 switches.

Examples from the ASCII Text Code

Code	Character
00110000	0
00110001	1
00110010	2
00110011	3
00110100	4
00110101	5
01000001	A
01000010	B
01000011	C
01000100	D
01000101	E

From Base10 to Base2

- Divide the value by 2 and put the remainder to the right most position
- $15 = 2*7 + 1$

  $= 2*(2*3+1)+1$

  $= 2*2*(2*1+1)+2*1+1$

$15 = 1111_b$

Assignment

Watching BBC

Clip 1

Clip 2