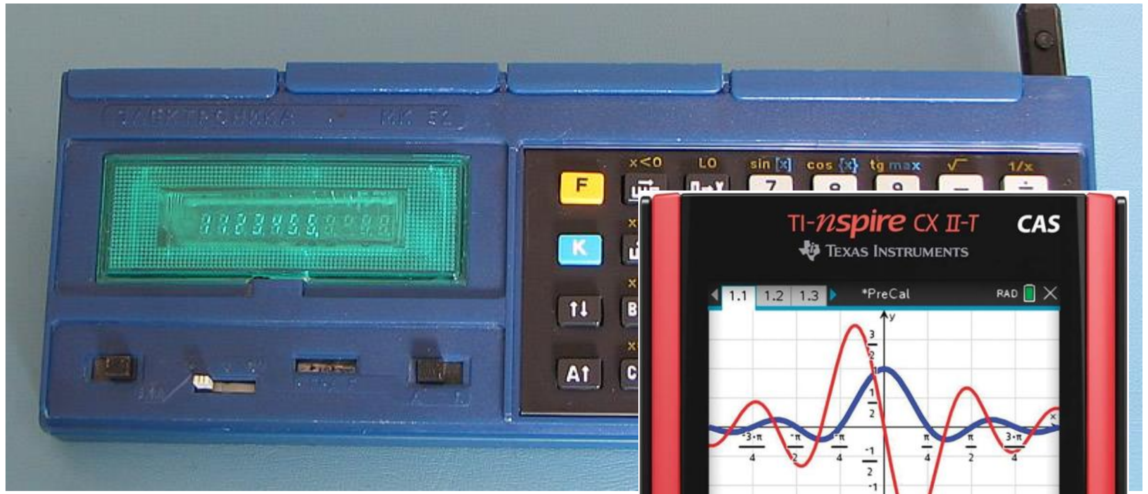
THE UNIVERSITY of TENNESSEE KNOXVILLE

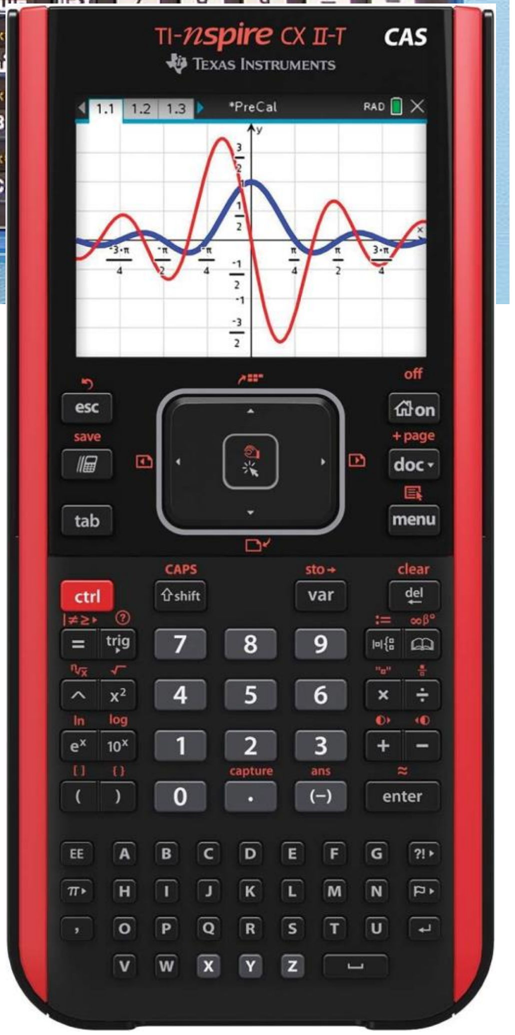
Lecture 02: Python Ecosystem and Numbers in Python

Sergei V. Kalinin

Numbers, Functions, Big Data

- **Numbers:** MatLab, Python, C++, …
- **Functions:** Mathematica, some Python libraries (SymPy)
- **Big Data:** Python, Julia, ….





How we can run code

- Google Colabs
- AWS SageMaker notebooks
- IDE: Spyder, PyCharm, etc.
- Command line interface

Key aspect of teaching now compared to 1.5 years ago: ChatGPT

- Focus on ideas more then code
- Use but verify (confabulations)
- Build foundation easier – but will have to learn advanced topics anyway

ChatGPT and coding

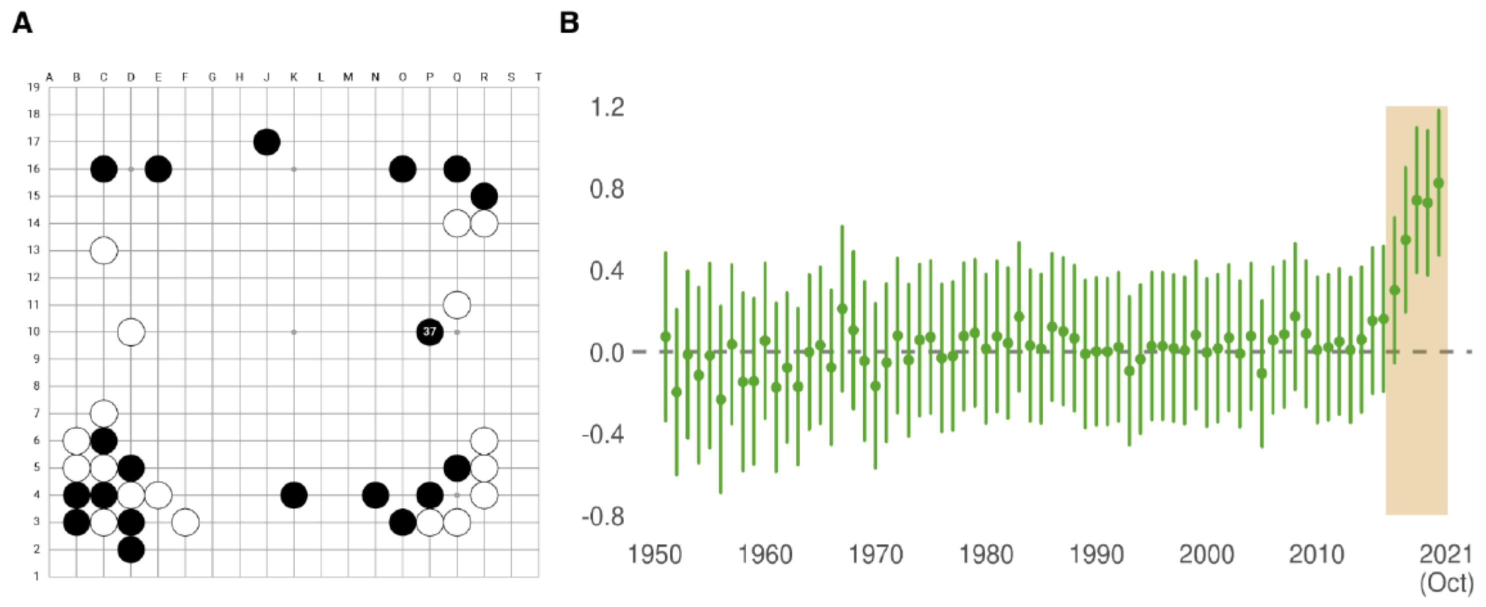
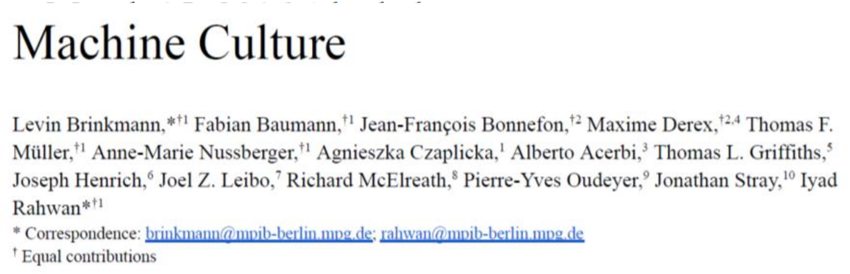


Figure 3: Go play before and after the introduction of AlphaGo. A AlphaGo, in its match against Go world champion Lee Sedol, made a highly unusual and strategic 37th move by placing its stone further from the edge, towards the center of the board, deviating from the traditional strategy of securing territory along the periphery during the early stages of the game. With this unconventional move, AlphaGo not only broke with centuries-old Go traditions but also paved the way for its ultimate victory in the match. **B** (reproduction based on [30]) Decision quality of professional Go players as evaluated by an algorithm performing at superhuman level. Decision quality significantly increased after Lee Sedol was beaten by AlphaGo



Machine Culture

Levin Brinkmann,[*†1] Fabian Baumann,[†1] Jean-François Bonnefon,[†2] Maxime Derex,[†2,4] Thomas F. Müller,[†1] Anne-Marie Nussberger,[†1] Agnieszka Czaplicka,[1] Alberto Acerbi,[3] Thomas L. Griffiths,[5] Joseph Henrich,[6] Joel Z. Leibo,[7] Richard McElreath,[8] Pierre-Yves Oudeyer,[9] Jonathan Stray,[10] Iyad Rahwan[*†1]

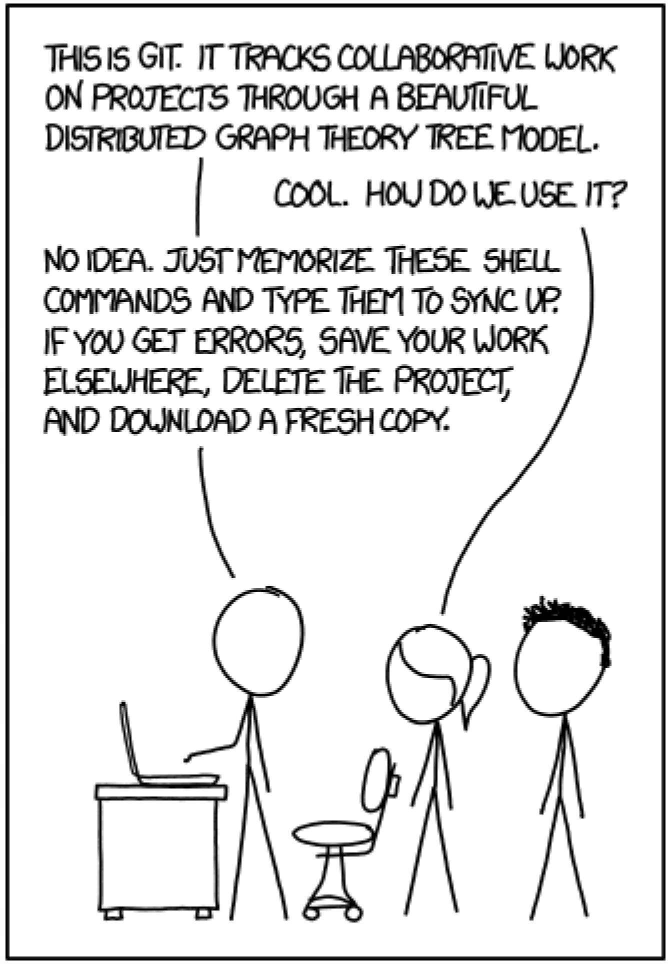
[*] Correspondence: brinkmann@mpib-berlin.mpg.de; rahwan@mpib-berlin.mpg.de

[†] Equal contributions

Code Repositories and Version Control

- Sharing scripts between users can be workable for immediate or short-term needs, but is not scalable nor lasting
- For reproducibility, it is better to have codes that reside in packages that are documented and well tested
- Most of you are familiar with python packages; but many are probably new to version control
- Version control systems such as git enable multiple people to work on a single software project at the same time to speed up development and ensure consistency
- Git is an open-source distributed version control system. It maintains a history of changes that have occurred in the project and allows for updates as well as reversions to older ‘commits’.

How we can share code:


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

Git would take a significant amount of time to explain in detail. However, there are plenty of online tutorials, e.g.

https://www.atlassian.com/git/tutorials

Let’s just have a look!

What language do we use:

Main Python libraries we will use:

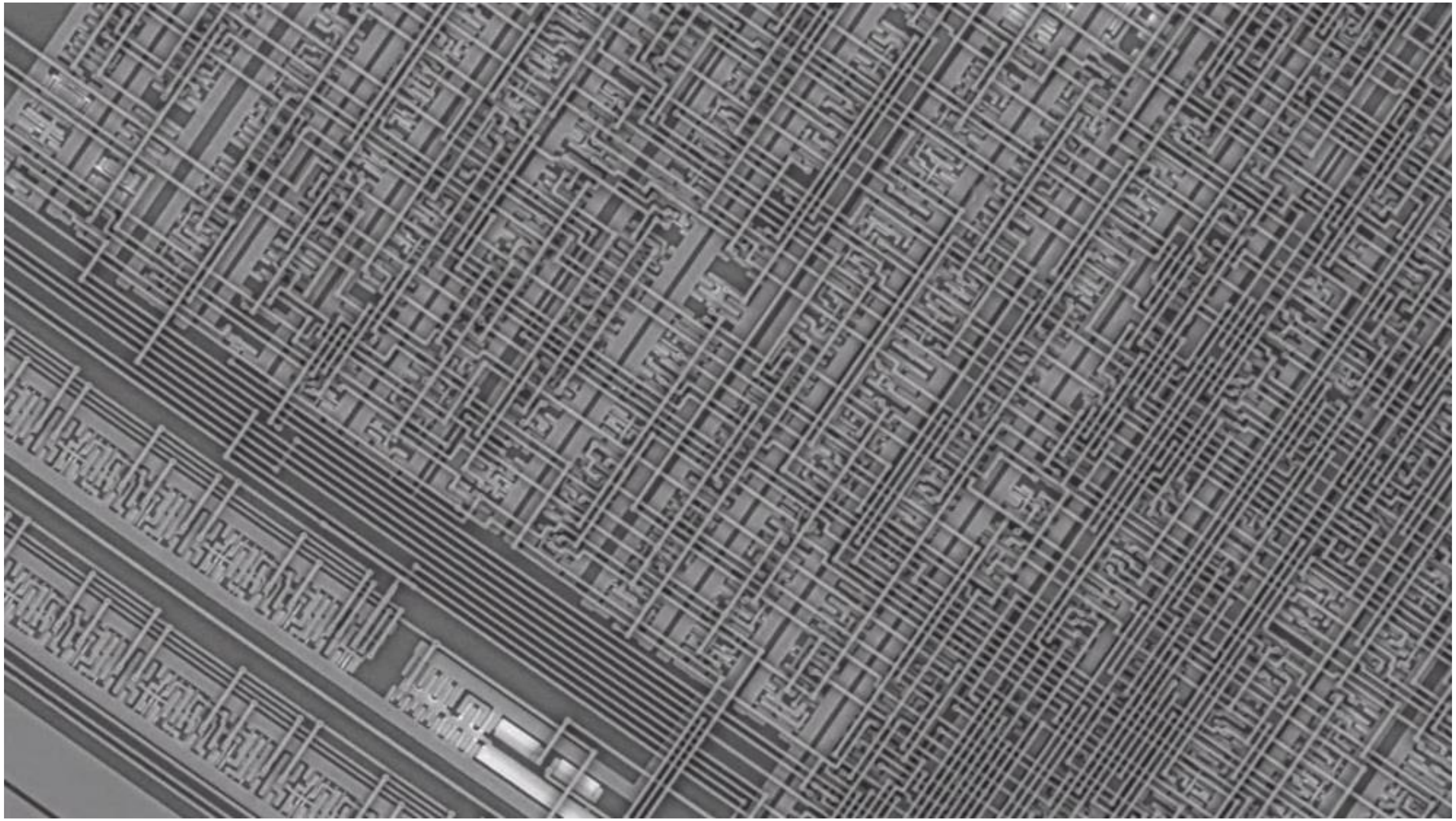
1. NumPy
2. MatPlotlib
3. Scikit-learn
4. Keras

Other libraries we may use:

1. Seaborn
2. GPax
3. SciPy

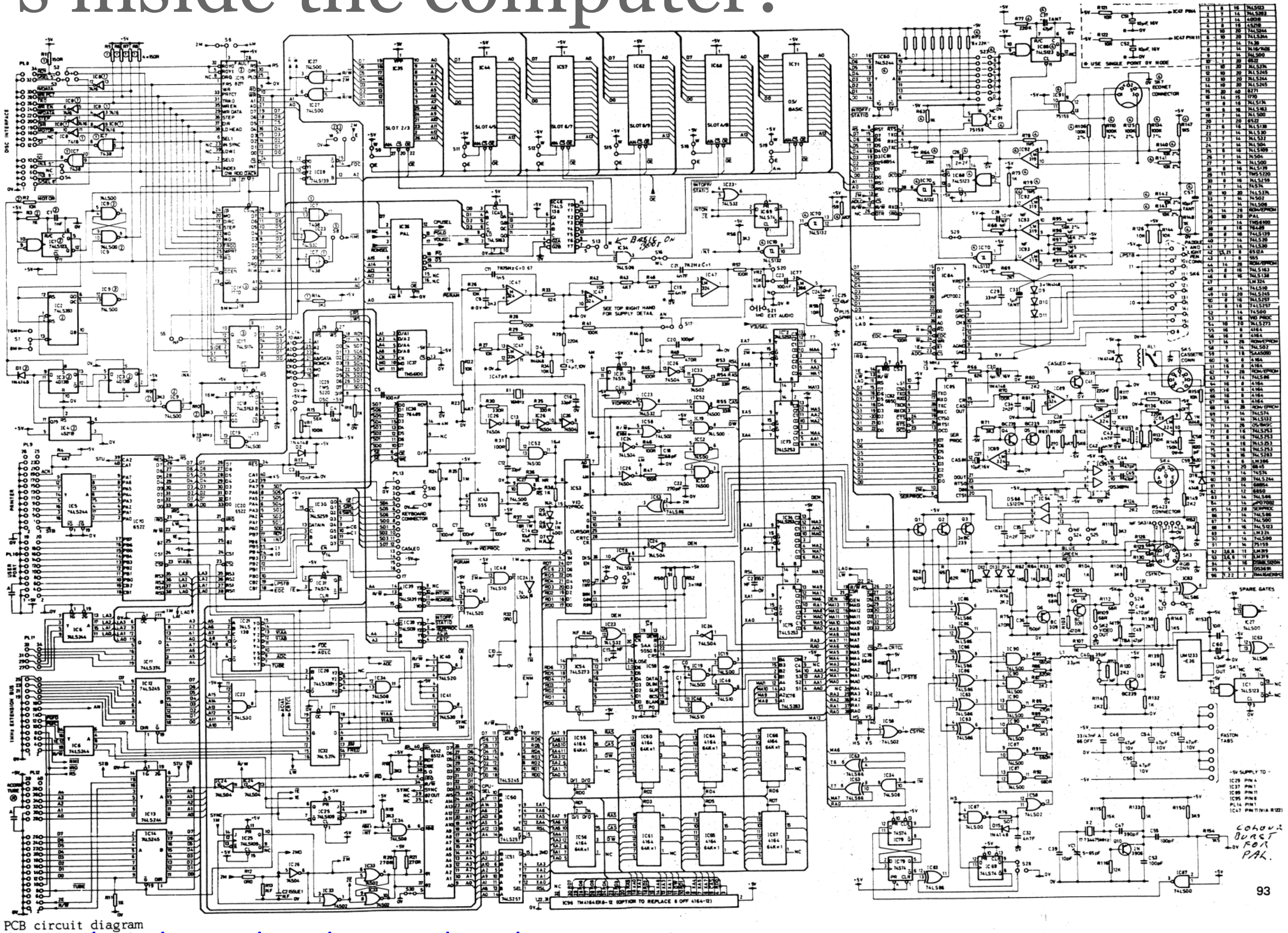
We will learn these as we need them!

What’s inside the computer?



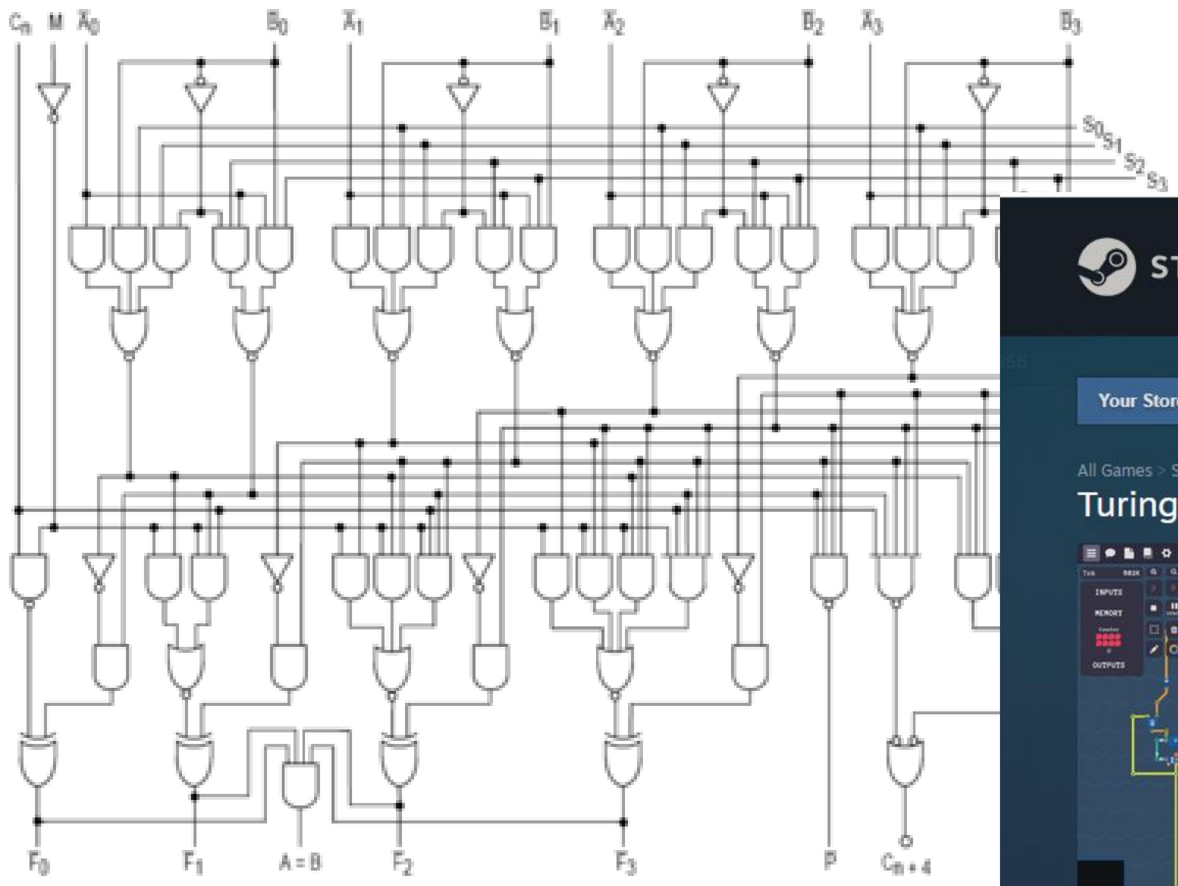
https://www.extremetech.com/extreme/191996-zoom-into-a-computer-chip-watch-this-video-to-fully-appreciate-just-how-magical-modern-microchips-are

What's inside the computer?

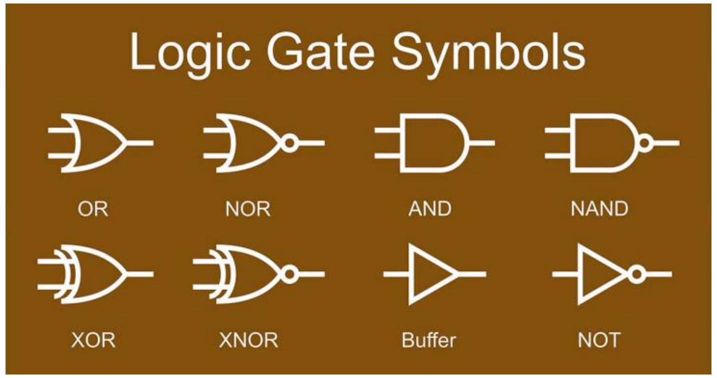


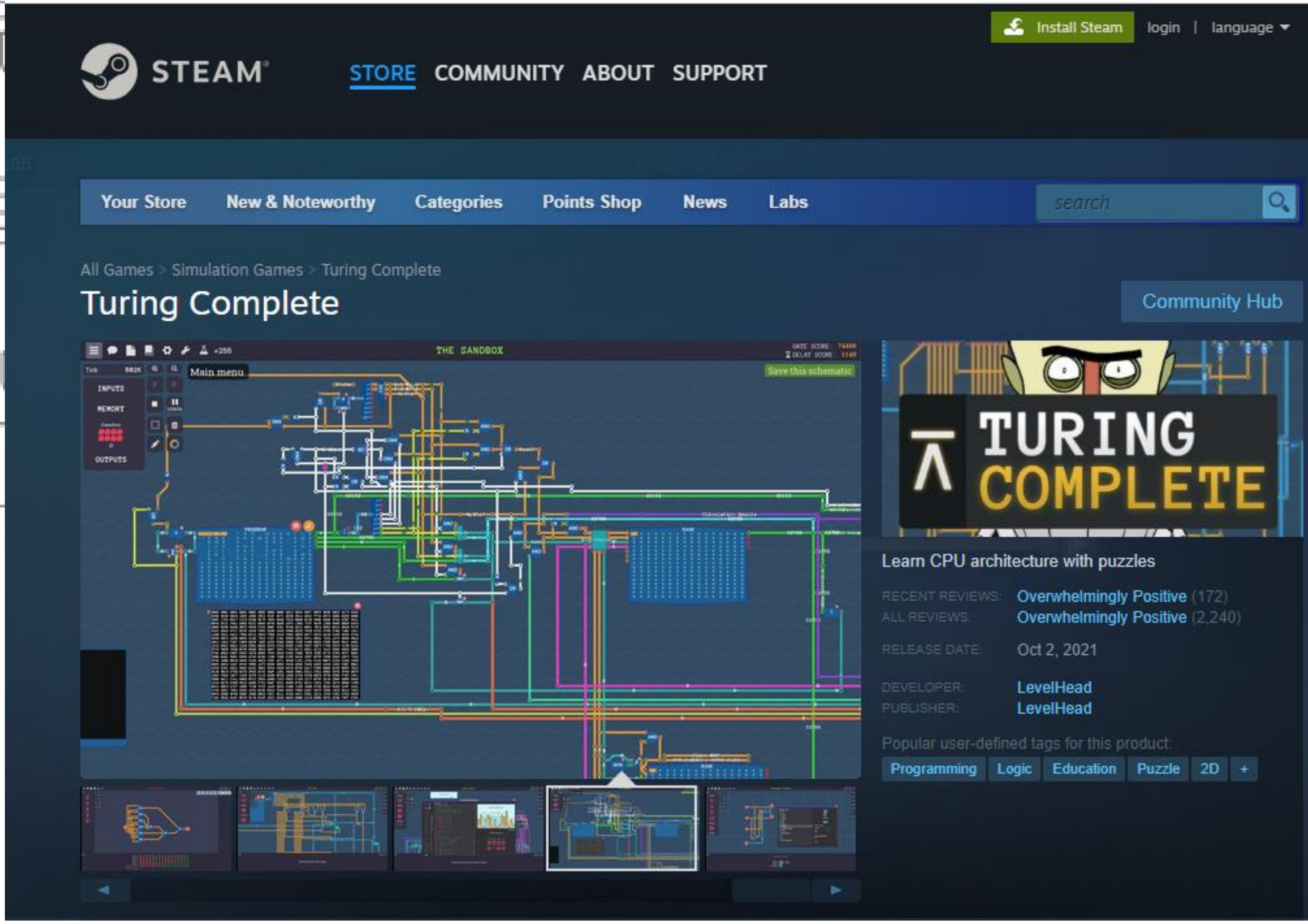
https://mdfs.net/Info/Comp/BBC/Circuits/BBC/bbcplus.gif

Binary logic!



https://en.wikipedia.org/wiki/74181





All Games > Simulation Games > Turing Complete

Turing Complete

Learn CPU architecture with puzzles

RECENT REVIEWS: Overwhelmingly Positive (172)
ALL REVIEWS: Overwhelmingly Positive (2,240)
RELEASE DATE: Oct 2, 2021
DEVELOPER: LevelHead
PUBLISHER: LevelHead

Popular user-defined tags for this product:
Programming Logic Education Puzzle 2D +

Numbers

Integer (int): Represents whole numbers, both positive and negative. Example: 5, -3, 42

Floating Point (float): Represents real numbers (numbers with a fractional part). Includes a decimal point. Example: 3.14, -0.001, 2.0

Complex Numbers (complex): Consists of a real and an imaginary part. The imaginary part is denoted by a 'j' or 'J'. Example: 3 + 4j, -1.5 + 2.5j

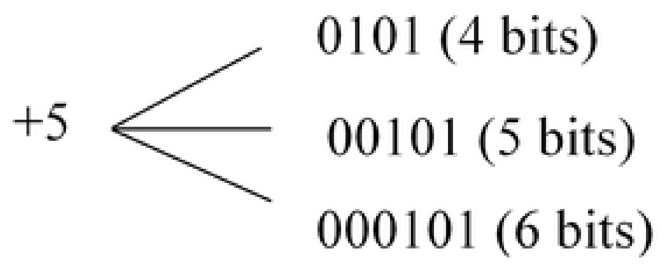
Binary: Represents numbers in base 2. Prefixed with 0b or 0B. Example: 0b1010 (equivalent to decimal 10)

Octal: Represents numbers in base 8. Prefixed with 0o or 0O (the letter 'o', not the number '0'). Example: 0o12 (equivalent to decimal 10)

Hexadecimal: Represents numbers in base 16. Prefixed with 0x or 0X. Uses digits from 0 to 9 and letters from A to F (or a to f). Example: 0xA (equivalent to decimal 10)

Integer numbers

Numbers on a computer are represented by bits



Most typical native formats:

- 32-bit integer, range –2,147,483,647 (-2^{31}) to +2,147,483,647 (2^{31})
- 64-bit integer, range ~ -10^{18} (-2^{63}) to $+10^{18}$ (2^{63})

Python supports natively larger numbers but calculations can become slow

Not all numbers can be fully represented!

A floating-point number in Python is composed of two parts: the mantissa (or significand) and the exponent, both of which are based on powers of two. The format is similar to scientific notation, where a number is represented as a * 2^b. Here, a is the mantissa, and b is the exponent.

Precision: Floating-point numbers are typically double precision (64-bit) following the IEEE 754 standard. This provides a significant degree of accuracy but can still lead to rounding errors in complex calculations.

Syntax: Floating-point numbers can be declared simply by including a decimal point. For example, 3.0, 4.2, -0.5. They can also be specified using scientific notation, e.g., 1.23e4 which is equivalent to 12300.0.

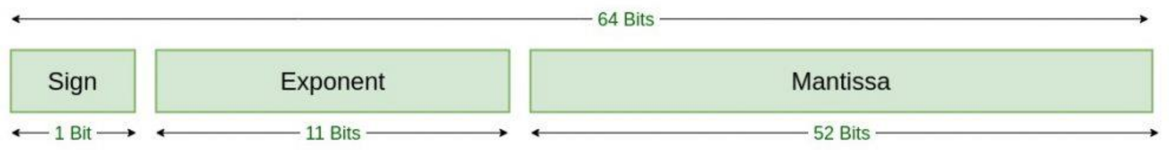
Limitations: Due to their binary nature, not all decimal fractions can be precisely represented. For instance, 0.1 in Python is an approximation, leading to potential precision errors in calculations.

Python uses a type of rounding to minimize this error, but it's important to be aware of it, especially in numerical computations.

Not all numbers can be fully represented!

Floating point numbers represented by bit sequences separated into:

- Sign S
- Exponent E
- Mantissa M (significant digits)



Double Precision
IEEE 754 Floating-Point Standard

$$x = S \times M \times 2^{E-e}$$

Main consequence: Floating-point numbers are not exact!

For example, with 52 bits one can store about 16 decimal digits

Range: from ~ -10^{308} to 10^{308} for a 64-bit float

There are workarounds

- **Creation of Rational Numbers:** You can create fractions from integers, floats, decimal numbers, or strings representing a fraction.
- **Arithmetic Operations:** The module supports basic arithmetic operations like addition, subtraction, multiplication, and division with fractions.
- **Maintaining Exactness:** Fractions are stored as two integers, representing the numerator and the denominator. This ensures exact arithmetic operations, unlike floating-point numbers where precision issues can arise.
- **Conversion and Simplification:** Fractions are automatically simplified. For example, fractions.Fraction(4, 6) will simplify to 2/3. You can also convert fractions to other numeric types like floats or decimals.

```
from fractions import Fraction

# Creating fractions
f1 = Fraction(3, 4)  # Fraction from two integers
f2 = Fraction('1/4')  # Fraction from a string
f3 = Fraction(0.5)    # Fraction from a float

# Arithmetic operations
sum_f = f1 + f2       # Adds 3/4 and 1/4
mul_f = f1 * f3       # Multiplies 3/4 and 1/2

print("Sum:", sum_f)  # Output: Sum: 1
print("Product:", mul_f)  # Output: Product: 3/8
```

On to Colabs!