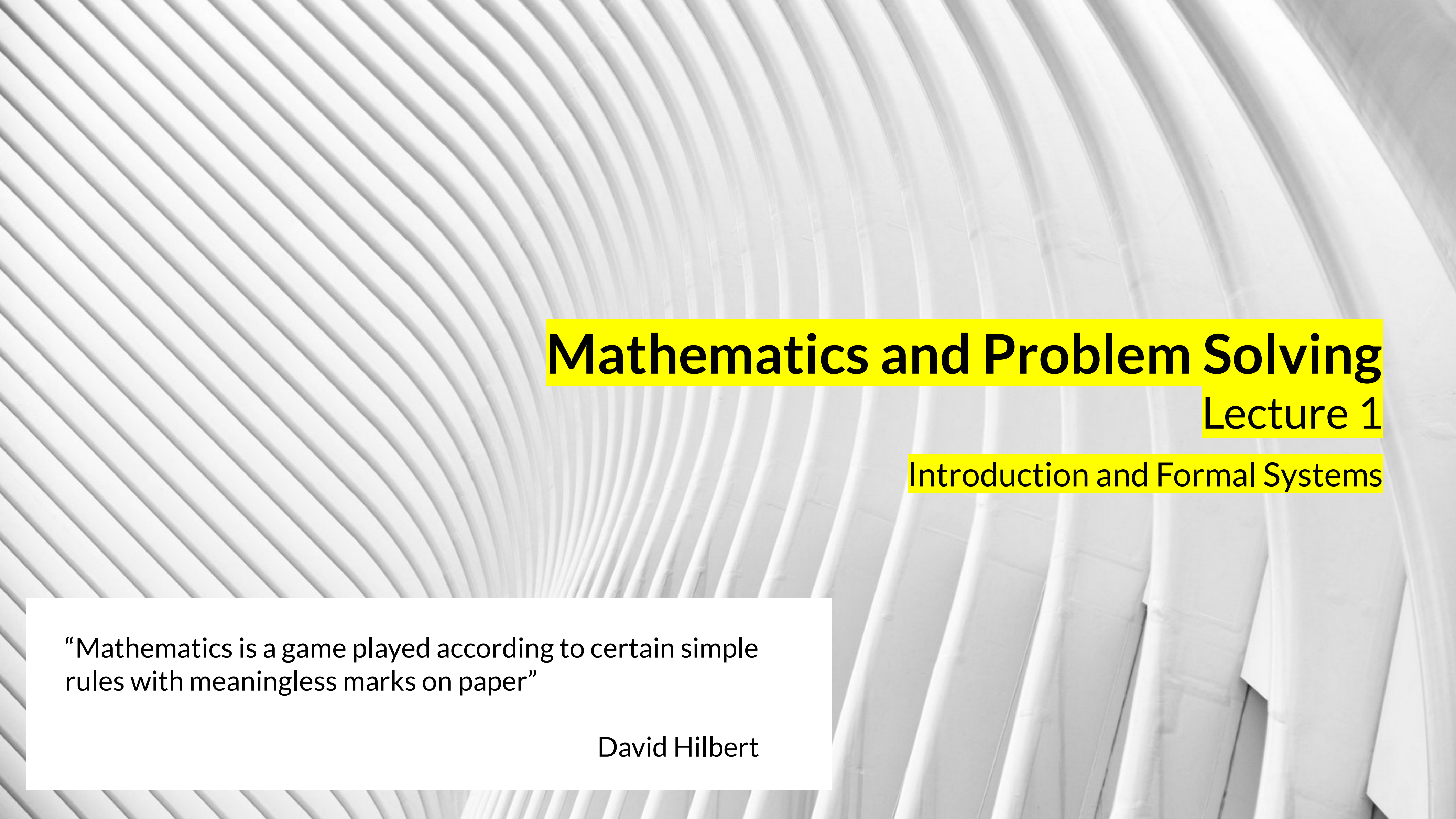


Mathematics and Problem Solving

Lecture 1

Introduction and Formal Systems

“Mathematics is a game played according to certain simple rules with meaningless marks on paper”

David Hilbert

Overview

- The Course
- Computation
- A bit of history
- Formal Systems
- Revising the Basics

The Course

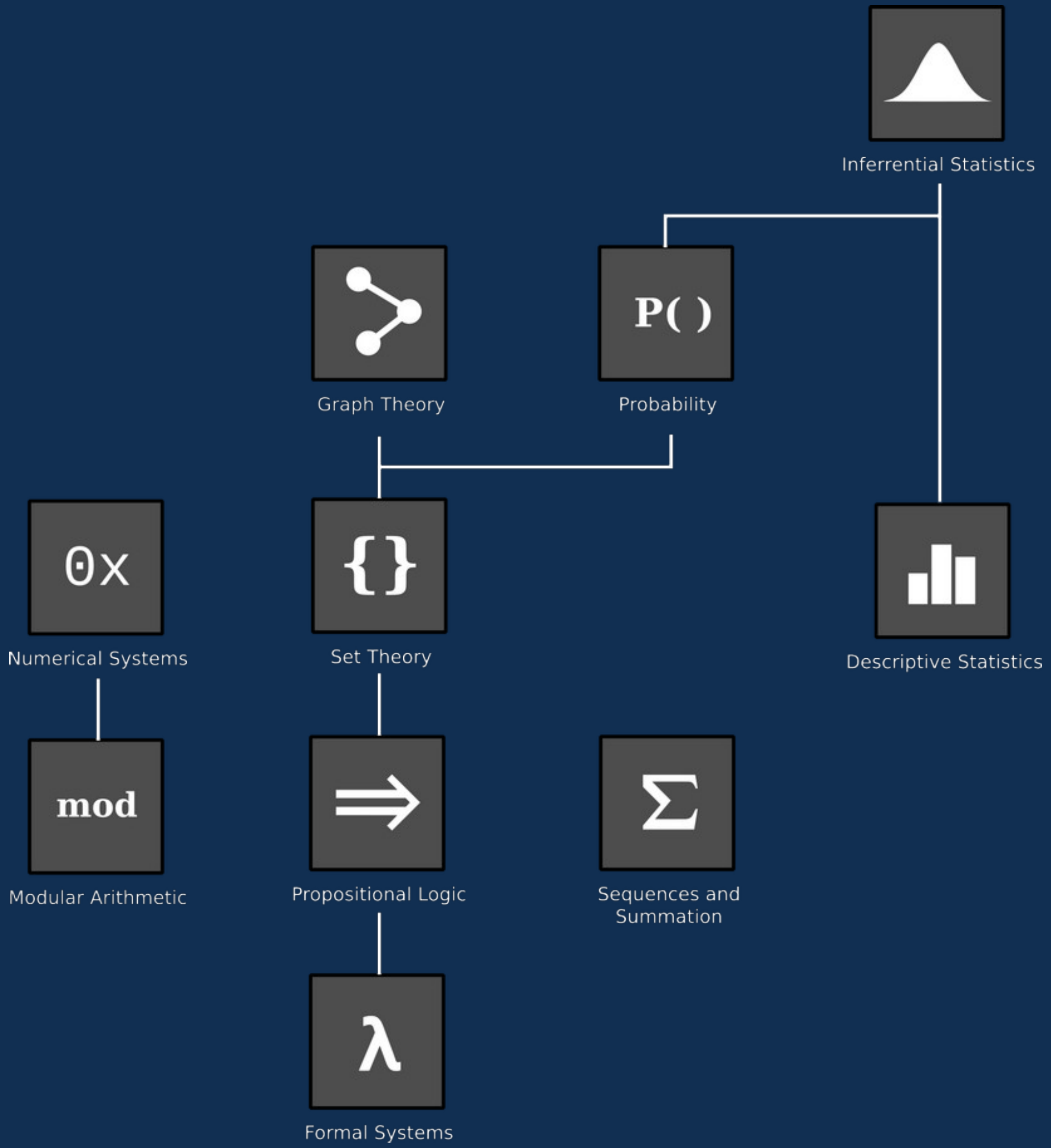
Structure

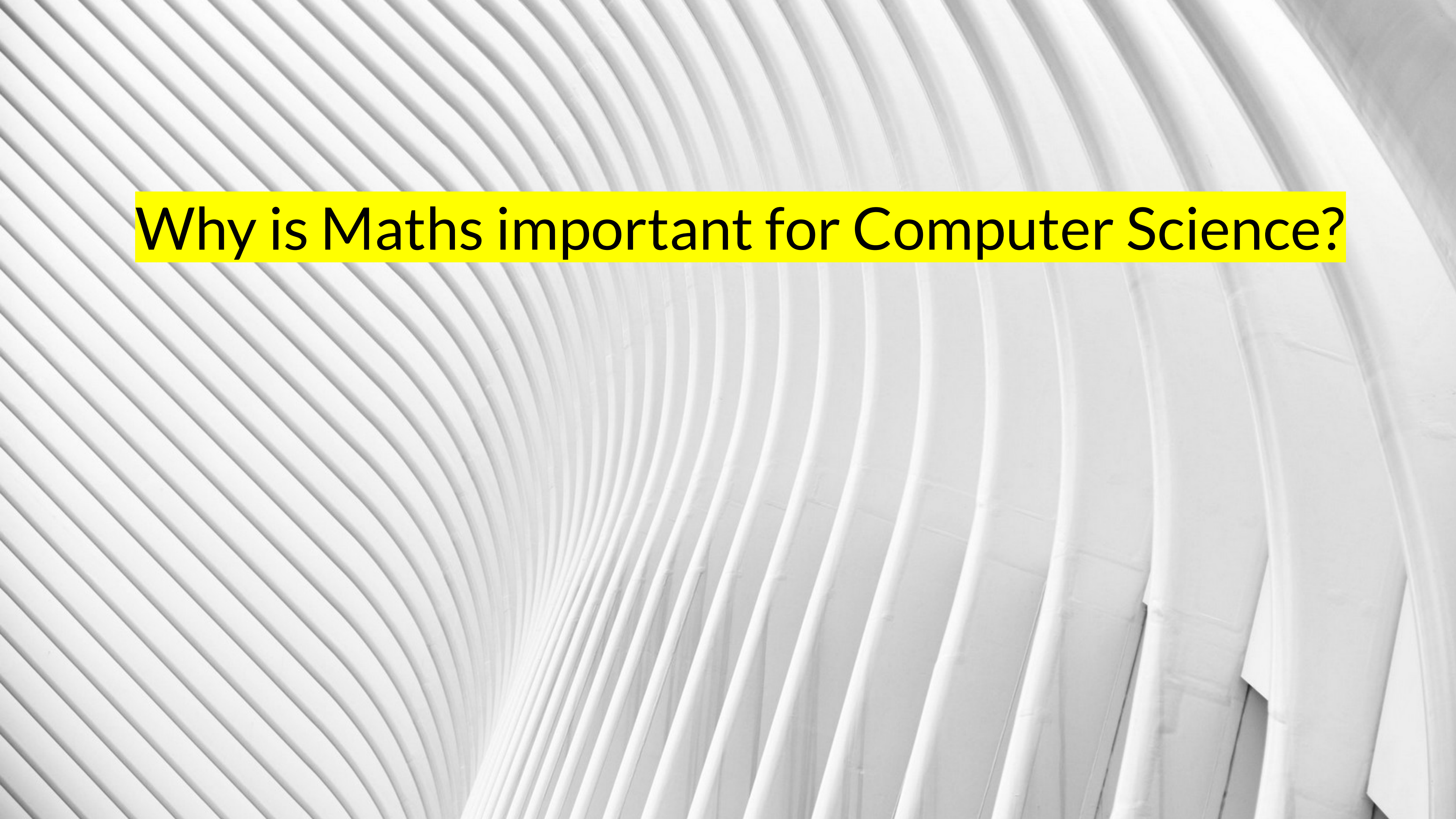
- Each week we have a 4 hour lecture/practical
  - I introduce some mathematical concepts
  - Slides are broken up by exercises
  - You'll see similar exercises in the assessment
- Problem questions each week
  - Practice the content
  - Submit them as your assessment

Assessment

- Each week there will be a sheet of problem questions
- You will submit all of these together at the end of the course
  - There will be submission instructions on Moodle closer to the deadline
  - Some variation on “staple them all together”
- Do them each week
  - It’s a lot of work to leave to the end
  - Get help on what you‘re not confident on
- It must be all your work

Content


Inferrential Statistics
Graph Theory
Probability
Numerical Systems
Set Theory
Descriptive Statistics
Modular Arithmetic
Propositional Logic
Sequences and Summation
Formal Systems



Why is Maths important for Computer Science?

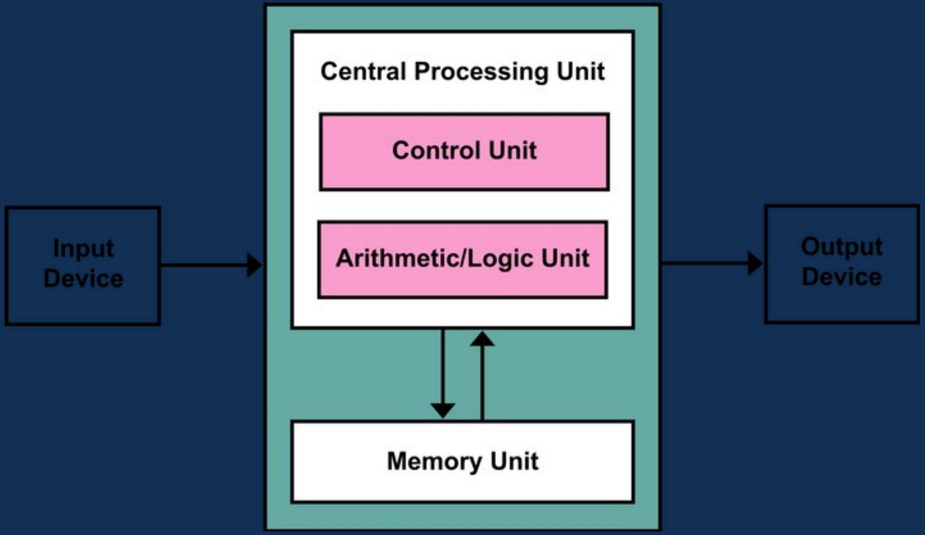
Computers

Exercise 1:

What is a computer?

Computers

- We usually think of computers as
  - Magical boxes with user-friendly interfaces
  - von Neumann architecture
- But that’s just what we’re used to



Human Computers

- For most of history, “computers” were human
- Calculations were broken down into steps and then performed by teams of people

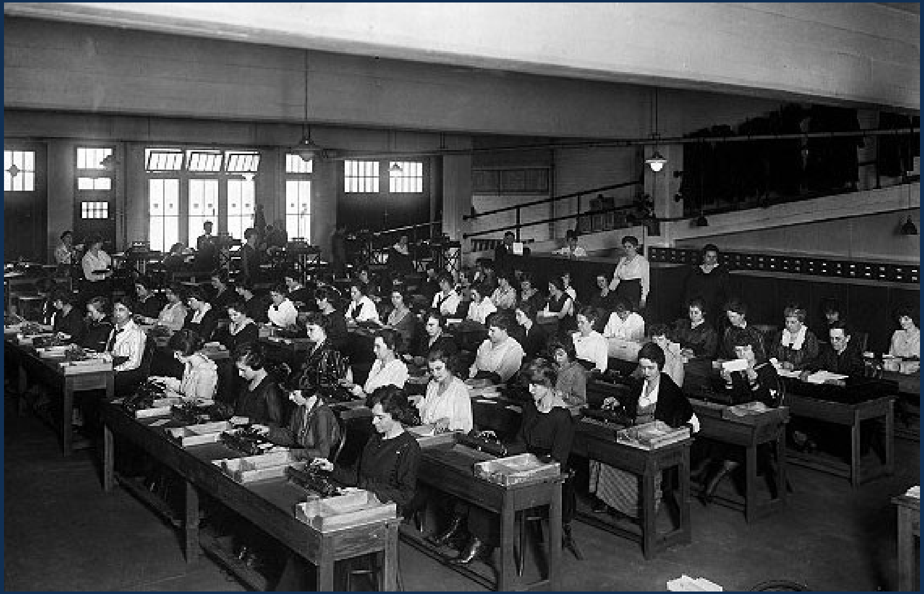
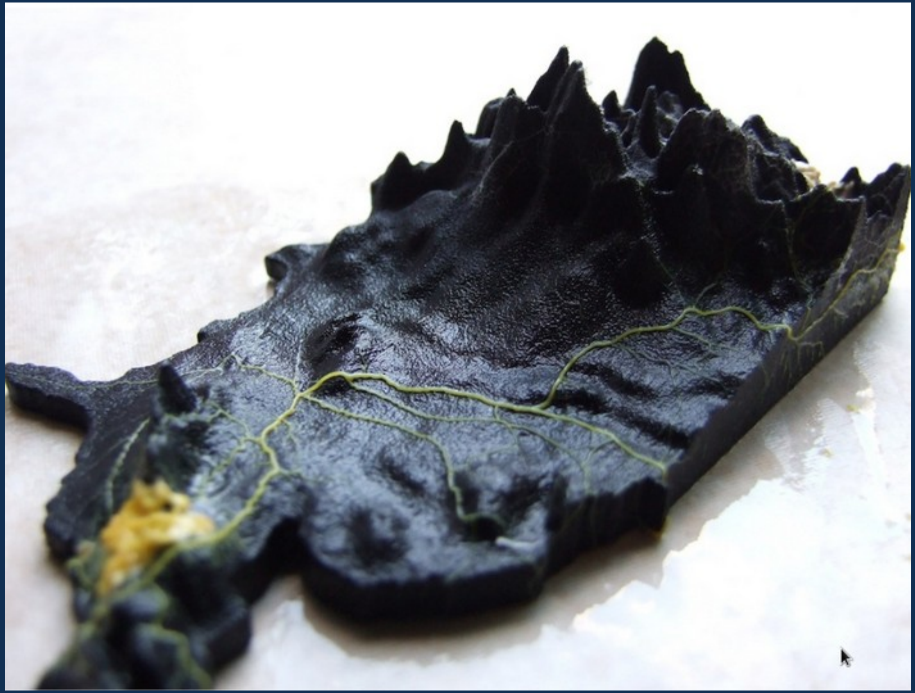


Image: Computer History Museum

Slime mould

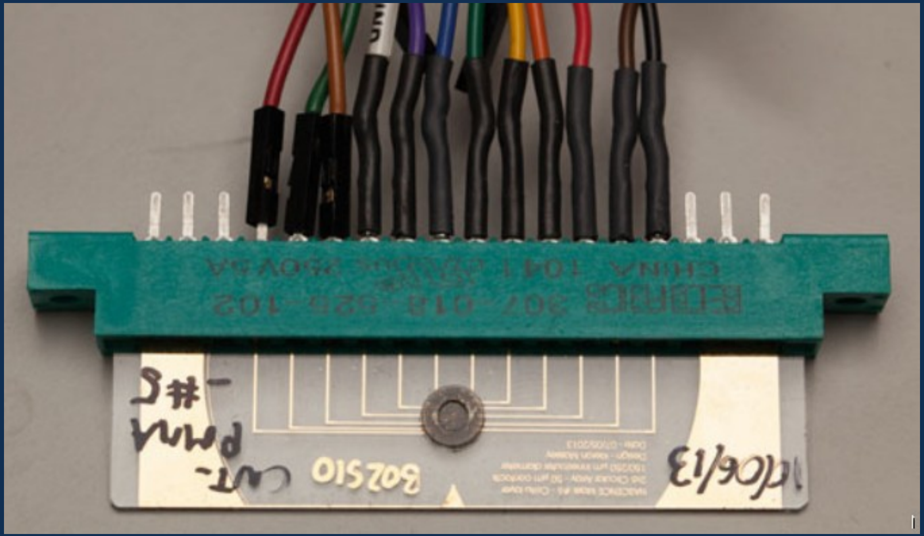
- Slime mould computes the shortest path, e.g.
  - Through a maze
  - A topographical map of the US



Adamatzky, Andrew I. "Route 20, autobahn 7, and slime mold: approximating the longest roads in USA and Germany with slime mold on 3-D terrains." IEEE transactions on cybernetics 44.1 (2013): 126-136.

Computation In-*Materio*

- The conductive properties of a lump of carbon nanotubes when peturbed by small currants
  - Some input (encoded into voltages)
  - Physical properties of the material *is* the computation
  - Some output (read off voltages)



Dale, Matthew, Julian F. Miller, and Susan Stepney. "Reservoir computing as a model for in-materio computing."Advances in Unconventional Computing. Springer, Cham, 2017. 533-571.

Computation

- The physical instantiation of the computation can vary
  - Transistors
  - Humans
  - Mould
  - Carbon Nanotubes
  - Buckets of water
  - The universe?

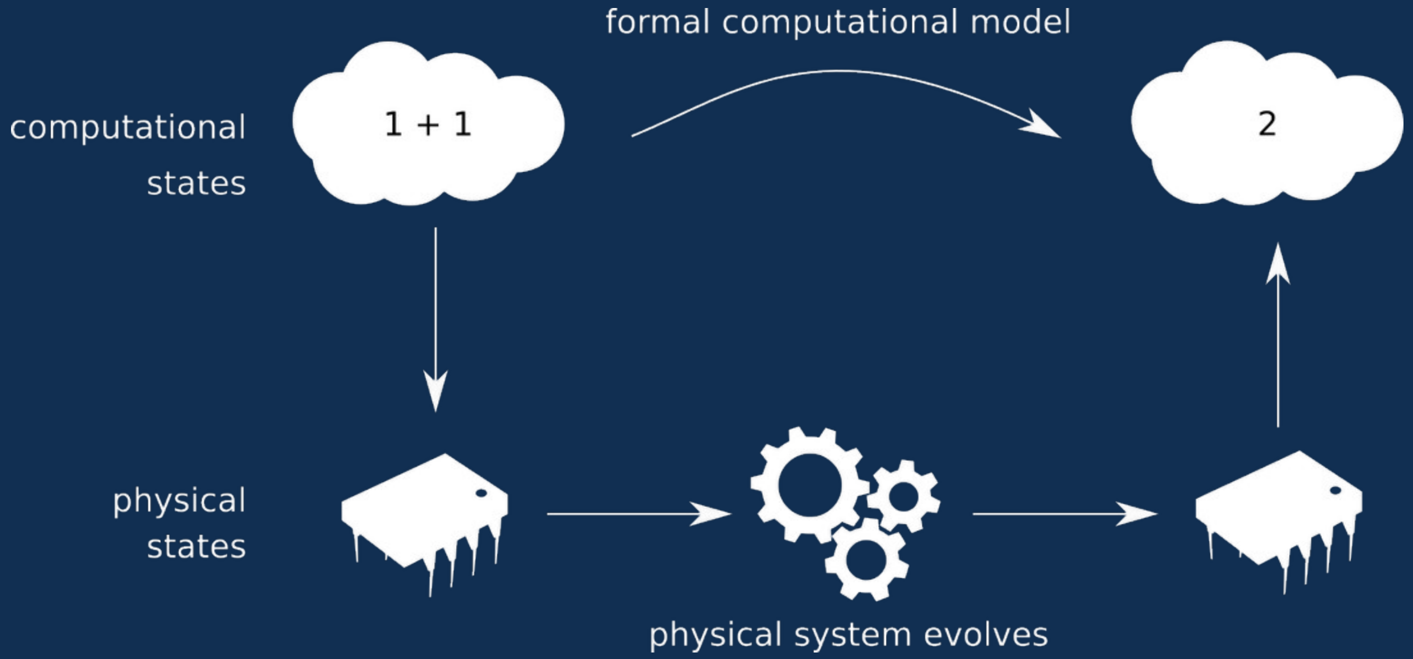
- If so, how do we understand computation?

  Exercise 3:

  What is computation?

The Mapping account of computation

- A physical system performs computation when there is a mapping between states of the system and computational states



Lambda Calculus

- Formal model of computation
  - Turing Complete
  - Can express any computation possible with a computer
- “Smallest universal programming language”
- Uses just strings of symbols and rules to rewrite them
  - All you need is a pencil and paper!
- You do not need to learn Lambda Calculus for this course!
  - We’re seeing this an example of a formal model of computation

Lambda Calculus

- In lambda calculus, everything is a function
  - Functions look like this

$$\lambda\, sz\,.\, s(z)$$

- When we want to compute something we
  - Translate it into a lambda calculus function
  - Apply the rules of lambda calculus
  - Translate back to see our answer

Lambda Calculus

- Define functions
  - λ <variable > . <expression>
- Function application
  - (λ <variable > . <expression>) <expression>
  - (λx . x) y
  - Rewrite occourances of x with y in λx . x
  - (λx . x) y = [y / x] x = y

Representation in Lambda Calculus

- Everything is represented by a function
- Zero

  λ s z . z
- Numbers 1-3:

  λ sz . s(z)

  λ sz . s(s(z))

  λ sz . s(s(s(z))
- True and False

  λxy.x

  λxy.y
- Identity function

  λ z . z
- Successor function

  λwyx . y (wyx)
- Multiplication

  λxyz . x (yz)
- And

  λxy.xy(λuv.v)
- Lots more can be defined...

Working with Lambda Calculus

- Let's say we want to calculate the sucessor of 1 (=2)
- We translate that into lambda calculus as
  - S ≡ λwyx . y (wyx)
  - 1 ≡ λ sz . s(z)
  - S1 ≡ (λwyx . y (wyx)) λ sz . s(z)
- Now we derive an answer
  - (λwyx . y (wyx)) λ sz . s(z)
  - λyx . y (λ sz . s(z) yx)
  - λyx . y (λ z . y(z) x)
  - λyx . y (y (x))
- Which we can see has the form
  - λ sz . s(s (z))
  - Which is 2!

Summary

- Computers perform computation by physically instantiating problems
  - We can also understand computation mathematically
- Lambda calculus represents and performs computation in a very different way to a microprocessor
  - It shows the underlying mathematical nature of computation
  - It was invented to study mathematical properties of computable functions
  - It has informed the development of functional programming languages like Haskel

The SPAMputer

The SPAMputer

I have designed a computer. It's a mechanical device that works with cards with holes punched in them. The holes look like

.

There are also some special shaped holes that I've represented with the letters

s, p, a, m

Today you'll have to work through the steps manually

– As I've had difficulties getting a grant to build it

You can do this with a hole punch and cards. But it'll probably be easier if you just write it out on paper.

SPAMputer Transformations

The computer takes in a card with holes punched in it (a piece of paper with writing on it)

- The “input string”

It outputs another card with holes punched in it.

- The “output string”

The output string is the same as the input string, with a single change, determined by a rule, such as:

$$\text{s}\cdot \rightsquigarrow \cdot\cdot$$

“If there is an instance of the substring ‘s·’ in the input string you can replace it with ‘··’”

SPAMputer Rules

Each of the rules opposite identify and replace a substring

- Read $\rightsquigarrow$ as “is replaced with”

Only one rule applies at a time

The top two rules are special. “__” stands in for everything else in the string

- Basically, you need to swap (a/m) each time you do

$$s \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \rightsquigarrow s \cdot$$

- These rules also have the highest priority

$$s \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \_\_\, a \rightsquigarrow s \cdot \_\_\, m$$

$$s \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \_\_\, m \rightsquigarrow s \cdot \_\_\, a$$

If neither of the above apply, then perform one of the following

$$s \cdot \rightsquigarrow \cdot \cdot$$

$$\cdot \cdot \rightsquigarrow \cdot p \cdot$$

$$\cdot \cdot p \rightsquigarrow \cdot p s$$

$$\cdot p \rightsquigarrow s$$

Example

Example

The following is a valid computation:

$$\cdot\cdot p s \cdot a$$

$$\cdot\cdot p \cdot\cdot a$$

$$\cdot p s \cdot\cdot a$$

$$s s \cdot\cdot a$$

$$s \cdot\cdot\cdot a$$

$$\cdot\cdot\cdot\cdot a$$

$$s \cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot \_\_ a \rightsquigarrow s \cdot \_\_ m$$

$$s \cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot \_\_ m \rightsquigarrow s \cdot \_\_ a$$

If neither of the above apply, then perform one of the following

$$s \cdot \rightsquigarrow \cdot\cdot$$

$$\cdot\cdot \rightsquigarrow \cdot p \cdot$$

$$\cdot\cdot p \rightsquigarrow \cdot p s$$

$$\cdot p \rightsquigarrow s$$

Discovering the SPAMputer

Exercise 2:

You start with the card:

· · · · · · · · p · · · · · · a

Run the SPAMputer until no more rules can be applied.

What is the result?

s············__a ⤳ s·__m

s············__m ⤳ s·__a

If neither of the above apply, then perform one of the following

s· ⤳ ··

·· ⤳ ·p·

··p ⤳ ·ps

·p ⤳ s

Using the SPAMputer

Exercise 3:

Compute the results for the following inputs

1. s · p s s · a
2. · · · · p · · p · m
3. · · p · · · · · · · · · · · · a

s············__a ⤳ s·__m

s············__m ⤳ s·__a

If neither of the above apply, then perform one of the following

s· ⤳ ··

·· ⤳ ·p·

··p ⤳ ·ps

·p ⤳ s

Explaining the SPAMputer

Exercise 4:

What does the SPAMputer do?

s············__a ⤳ s·__m

s············__m ⤳ s·__a

If neither of the above apply, then perform one of the following

s· ⤳ ··

·· ⤳ ·p·

··p ⤳ ·ps

·p ⤳ s

Explaining the SPAMputer

Let me explain how to translate questions for the SPAMputer. Take a question like

“What is two hours after four pm”

Follow the rules opposite to get

$$\cdot\,\cdot\, p \,\cdot\,\cdot\,\cdot\,\cdot\, a$$

Run the SPAMputer for an answer

$$\cdot\,\cdot\,\cdot\,\cdot\,\cdot\,\cdot\, a$$

Which we translate as “six pm”

one $\rightsquigarrow$	$\cdot$
two $\rightsquigarrow$	$\cdot\,\cdot$
three $\rightsquigarrow$	$\cdot\,\cdot\,\cdot$
… etc.	
what is $\rightsquigarrow$	
hours after $\rightsquigarrow$	p
am $\rightsquigarrow$	m
pm $\rightsquigarrow$	a

Explaining the SPAMputer

Exercise 5:

Use the SPAMputer to answer the following questions

1. What is ten hours after three am?
2. What is one hours after two hours after three hours after four pm?

one $\rightsquigarrow \cdot$

two $\rightsquigarrow \cdot\cdot$

three $\rightsquigarrow \cdot\cdot\cdot$

... etc.

what is $\rightsquigarrow$

hours after $\rightsquigarrow$ p

am $\rightsquigarrow$ m

pm $\rightsquigarrow$ a

Explaining the SPAMputer

Exercise 6:

Answer the following:

1. What computation is the SPAMputer doing?
2. What numbers *can't* we represent in the SPAMputer?
3. Is there a more familiar way of representing the same computation?
4. Can we ask other questions like "what is *x* hours before *y*?"

   or "what is *x* lots of *y* hours?"

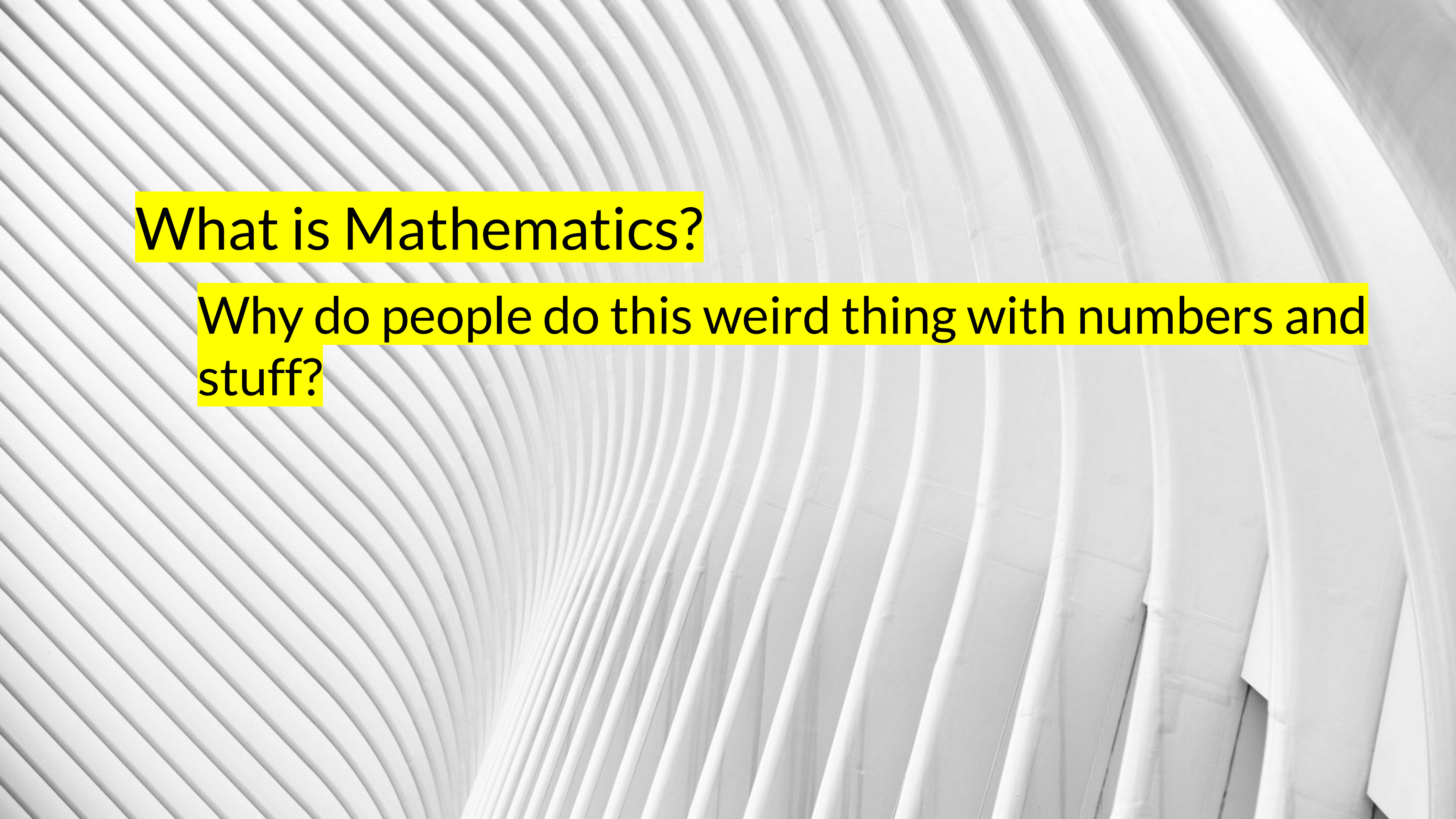
   Can the SPAMputer be extended to make these work?

Understanding Computation

- You don’t need mathematics to use computers
  - But to understand computation, you need mathematics

Computation is maths

- Maths isn't just numbers
  - I gave a formal definiton of the SPAMputer without using any numbers
- Maths isn't just a set of "platonic forms"
  - I *constructed* a set of rules that *constructed* the strings we get in the SPAMputer
  - Does the set of all possible SPAMputer strings exist?
- Maths is computation – a very "formal" kind
  - It's a process by which we *do stuff*
  - Makes it really good for abstracting and reasoning about physical computation
- To convince you, let's take a quick tour of the history of mathematics



What is Mathematics?

Why do people do this weird thing with numbers and stuff?

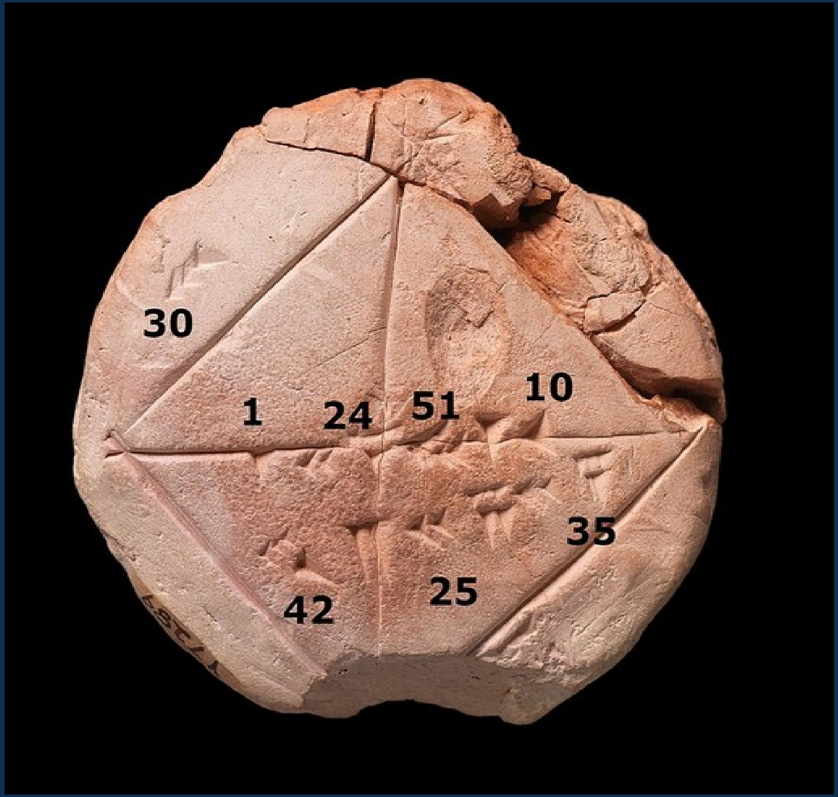
Why was it invented?

- Problem solving
  - Area of fields
  - Inheritance
  - Taxes
  - Distribution of grain

Babylonians

- Algorithmic
  - i.e. they came up with “programs” to solve common problems
- Base 60 number system
  - Zero as a placeholder
- Calendars

► Approximation of the square root of 2 in four sexagesimal figures, 1 24 51 10, The tablet also gives an example where one side of the square is 30, and the resulting diagonal is 42 25 35 (Wikipedia)



Indians

- Calculating very big numbers
  - Astronomy
  - Religious reasons
- Positional notation
  - Efficient at writing big numbers
  - Decimal number system
- Oral transmission in verse
  - Not like the way we write maths today!

Aryabhata

Greeks

- Lots of fledgling approaches to understanding the world competing
- Thought it was important to prove things
  - Mathematical proof from axioms
  - Legal evidence and proof
- Mathematicians needed to convince people that maths was a good idea

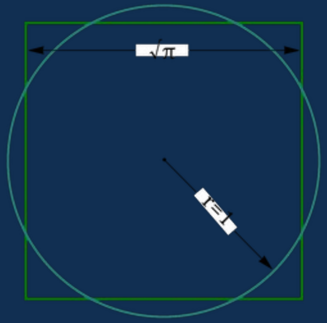
Pythagoras

Problems with proofs

- Say you’re a mathematician in ancient Greece, trying to justify why maths is a good way to understand the world
- Problem is, it’s easy to make embarassing mistakes, like arguing:
  - A super-right triangle is a triangle formed of 3 right angles.
  - The angles of a super right triangle add up to more than 180 degrees
  - The angles of some triangles add up to more than 180 degrees
- What’s wrong with this?
- How can you ensure that you won’t make this sort of mistake?

Constructions

- Provide a way of practically testing claims through constructions
- Constructivism is a philosophy of mathematics that says
  - to prove a mathematical object exists you must first give a way to make one
- For the Greeks, this meant constructing geometric shapes from a straight edge and a pair of compasses
  - e.g. How to “Square the Circle”
    - Constructing a square with the same areas as a circle
- Their maths wasn’t formulas, but diagrams!



Mathematics as Empirical

- Euclid expressed the axioms of his geometry as something you can *test*
  1. "Given two points, we can always draw a (unique) straight line segment connecting these points."
  2. "Given a straight line segment, we can (uniquely) extend the line segment indefinitely."
  3. "We can draw a (unique) circle with any desired center and radius."
  4. "All right angles are equal to one another."
  5. "If a straight line falling on two straight lines makes the interior angles on the same side less than two right angles, the two straight lines, if produced indefinitely, meet on that side on which are the angles less than the two right angles."

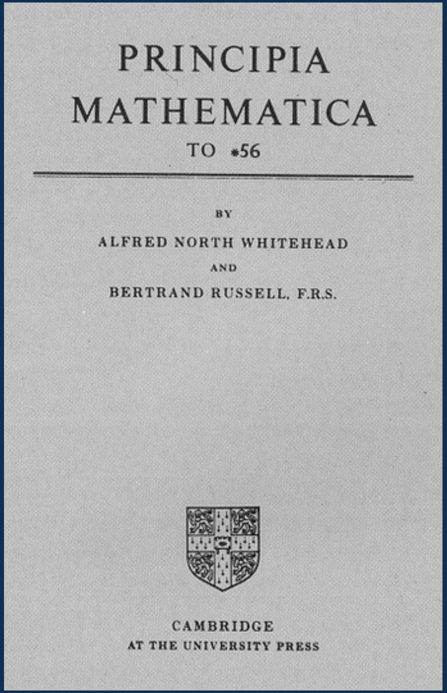
Islamic Empire

- Algorithms
- Algebra
  - “Reunion of broken parts”
- Still written in prose
  - Very few symbols before the 17th century!

Muhammad ibn Musa al-Khwarizmi

20th Century

- Generalisation, abstraction and formalisation
  - Constructions out of fashion
- Rise of mathematical logic
- Rather than proving as “self evident truth”, the goal became systems that are
  - Consistent
  - Complete (until Gödel’s Incompleteness Theorem!)
- Including attempting to build complete systems of all of mathematics
  - Founded upon Set Theory



PRINCIPIA MATHEMATICA

TO ∗56

BY

ALFRED NORTH WHITEHEAD

AND

BERTRAND RUSSELL, F.R.S.

CAMBRIDGE

AT THE UNIVERSITY PRESS

So what is Mathematics?

- Maths is a tool to do things
  - Measuring a field, tracking the seasons
  - Algorithms?
- Maths is about constructing things
  - shapes, numbers, proofs
  - Formal Systems?
- Maths is a language to abstract ideas about
  - logic, shape, quantity, arrangement
  - Computation?

Mathematical Language

- Way of expressing problems and their solutions
  - Verse, Prose, Diagrams, Symbols
- Nowadays we benefit from symbolic systems that have been invented
  - But just like the SPAMputer’s representations, they’re more or less arbitrary
  - Not “better” than using diagrams / prose
  - Maths is not just equations!
- Formal mathematics helps us communicate knowledge

What sort of knowledge?

- Games
  - Graphics/shaders
  - Path-finding
  - Physics engines
  - Simulations
  - Probability
  - Game Theory
  - …
- Software Engineering
  - Networks
  - Machine Learning
  - Algorithms
  - Data analysis
  - Type systems in programming languages
  - …
- Computer Science
  - Complexity
  - Models of computation
  - Computability Theory
  - Data structures
  - Cryptography
  - Type Theory

    …

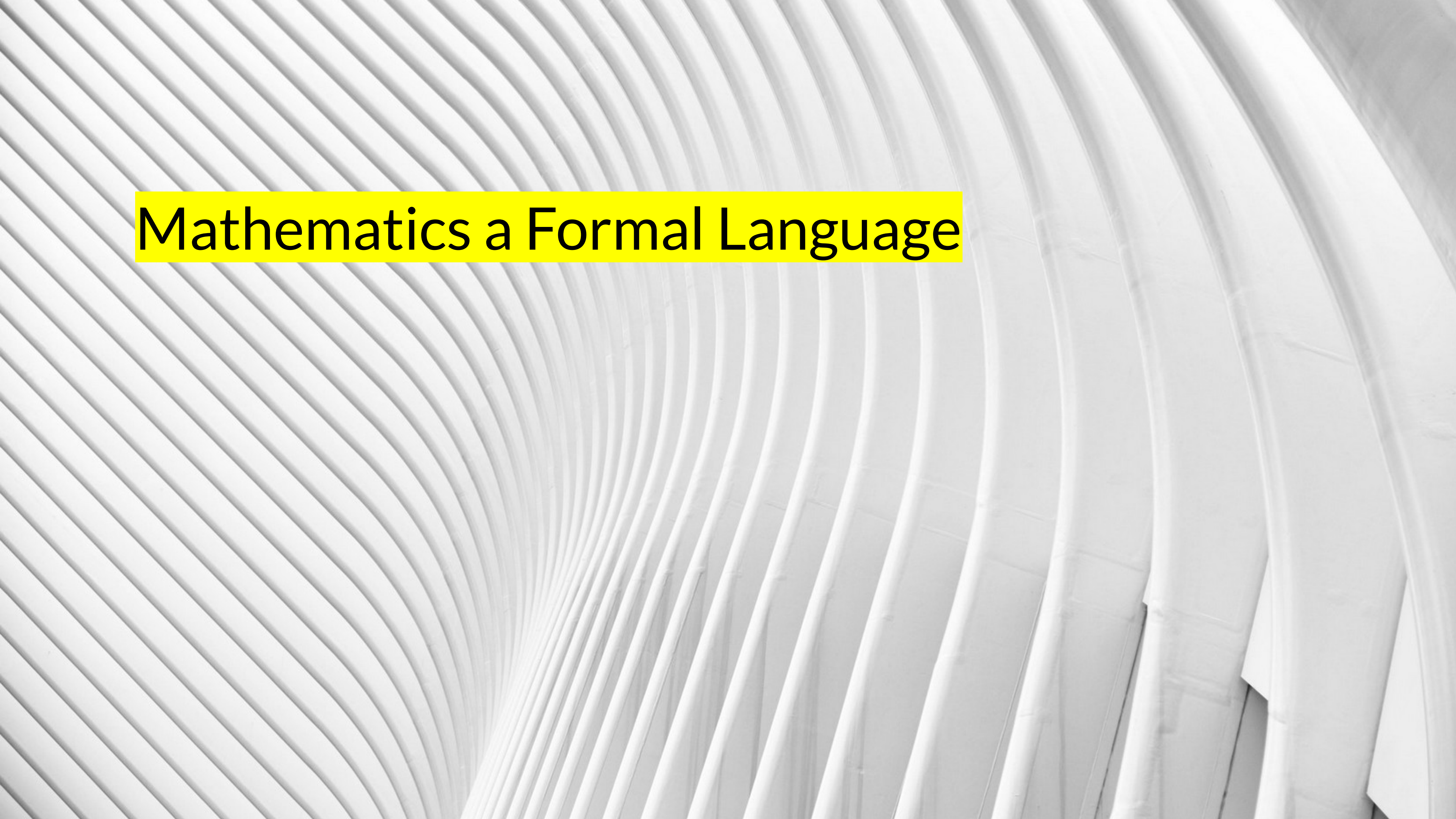
Why Maths?

- Why not just program the...
  - Shader, ML algorithm, cryptographic blockchain

  using your own intutition, or code from StackOverflow?

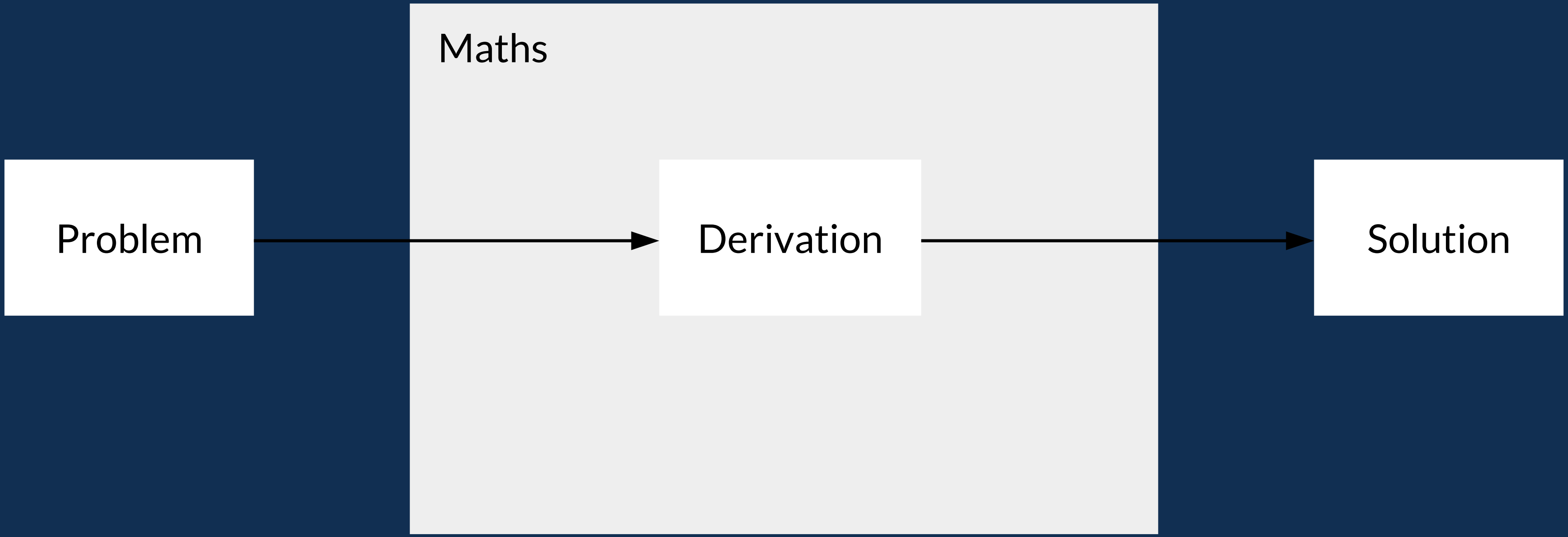
  1) You are doing maths, whether you realise it or not
  2) Mathematical language is
     - Precise
     - Unambiguous
     - Concise

     Perfect if you want to implement an algorithm and avoid mistakes in interpretation



Mathematics a Formal Language

Maths as Abstraction



Translation into Maths

- “My field should be 100m^2, but the markers have been washed away. I know it’s 10m long. How wide should it be?”

  Maths:

  $$100 = 10w$$

  Solve for w

Derivation

- Problem
  - 100 = 10w, solve for w.
- Derivation

  100 = 10w

  10 = w [Divide both sides by 10]

  w = 10 [Equality is symmetric]

Translation out of Maths

- “My field should be 100m^2, but the markers have been washed away. I know it’s 10m long. How wide should it be?”

$$100 = 10w$$

$$w = 10$$

- The width of my field should be 10m

Question:

Is this correct?

Abstraction

- Take only the things we think are essential
- Represent a complex system in terms of simple concepts
- Separate ideas from physical objects

Maths as a Formal System

- Maths is a Formal System (like our SPAMputer)
  - Abstracts the real world to a set of axioms (strings)
  - Uses laws to derive theorems (string rewriting rules)
  - Translates back into the real world
- All mathematical truth is relative to the truth of its axioms
  - Our solution of $w = 10$ was *true* (given arithmetic laws)

String Rewriting Systems

Formal Systems

- Formal systems represent
  - States (=Data Structures)
  - Rules that govern what transitions are allowed between states
- Imagine a game of chess
  - A data structure represents all the information about the board – the game state
  - We have rules that say what moves are allowed – what transitions are allowed between states

Transitions

- Imagine we had a state that we call X
  - X represents one configuration of a chess board
  - Or one state of a computer
- We have some rules R
  - The rules of chess
  - The rules of computation
- If a transition from X to another state Y is allowed under R, we can write

$$X \Rightarrow_R Y$$

Derivation

- A finite sequence of steps is called a **derivation**

$$X_1 \Rightarrow_R X_2 \Rightarrow_R \ldots \Rightarrow_R X_n$$

- We can write this

$$X_1 \Rightarrow^*_R X_n$$

(it is possible to go from X_1 to X_n following rules R in a finite sequence of steps)

- The **length** of a derivation is the number of steps it requires. Here it is *n*-1

Exercise 7:

In a formal system with rules *R*, and states *A*, *B*, *C*, *D*, *E* the following transitions are allowed

- $A \Rightarrow_R B$
- $A \Rightarrow_R C$
- $C \Rightarrow_R D$
- $D \Rightarrow_R E$

Is there a derivation $A \Rightarrow^*_R E$ in this system?

String Rewriting Systems

- A string rewriting system is a type of formal system
- Each state is represented by a string
  - Just like in our SPAMputer

    · · p s · a

  - Or in lambda calculus

    λ sz . s(z)

- Rules to transition between states (strings) are given by a grammar
- For example,
  - Say you are allowed to replace any single occourance of a given string with another string, e.g.

$$a \rightsquigarrow ab$$

  - Then

$$a \Rightarrow ab \Rightarrow abb \Rightarrow abbb \Rightarrow abbbb \Rightarrow \ldots \text{ etc.}$$

Exercise 8:

You have the string rewriting rules

$$X \rightsquigarrow ab$$

$$X \rightsquigarrow aXb$$

1. Starting with the string X, derive 5 strings in this system
2. Is it possible to derive the string bXa?

The MU Puzzle

Exercise 9:

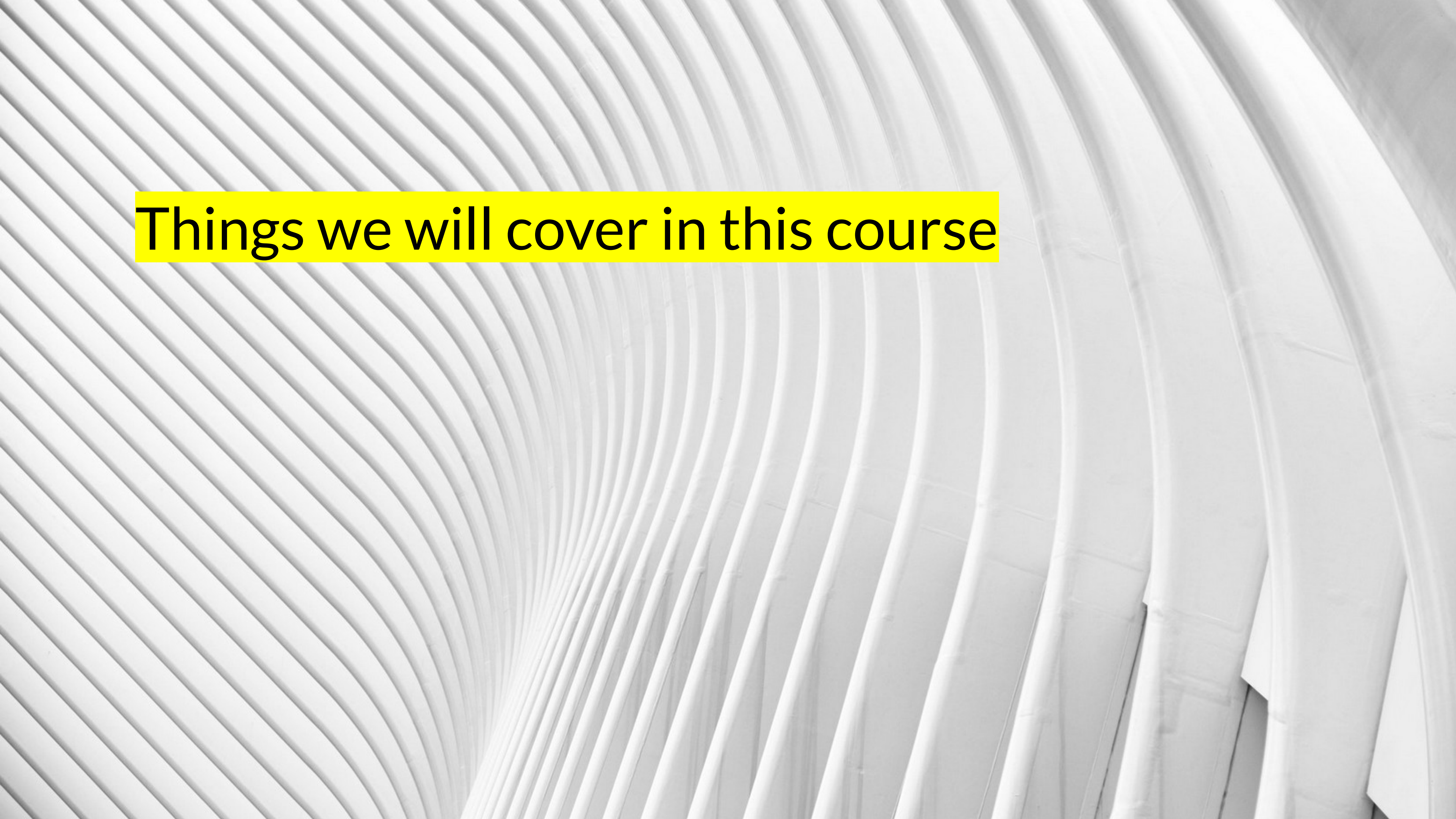
You have a collection of strings. At first you only have the string MI

These are the derivational rules:

- If a string ends in I, you can add U on the end, e.g. MI → MIU
- If you have a string of form Mx, you can have the new string Mxx
- If the substring III appears anywhere you can replace it for U
- If the string UU, you can replace it with an empty string

Is it possible to add the string MU?

From the book *Godel, Escher, Bach: An Eternal Golden Braid* by Douglas Hofstadter.



Things we will cover in this course

Beyond Arithmetic

- We're used to maths being about numbers

$$1 + 1 = 2$$

- That's arithmetic. It's a formal language.
  - There are all sorts of rules by which we can derive new statements in arithmetic
- We'll see new formal languages (a notation and system of rules) to describe
  - Logic
  - Sets (i.e. collections of things)
  - Networks (graphs)

New Concepts

- We want to be able to work with concepts such as
  - Binary numbers and operations
  - Algorithms
  - Networks
  - Logic
  - Databases
- We’re going to need to go beyond just representing numbers
  - (But we’ll need them, too)

Numbers, like 4

- What numbers exist?
  - Classes of Numbers
- How do we write them down?
  - Numerical Systems
- How do we add them up?
  - Algebra
  - Binary Arithmetic
  - Modular Arithmetic

Sequences

- What do we get if we put numbers in an order?
  - Sequences
- How do we add a series of numbers?
  - Summation

Logic

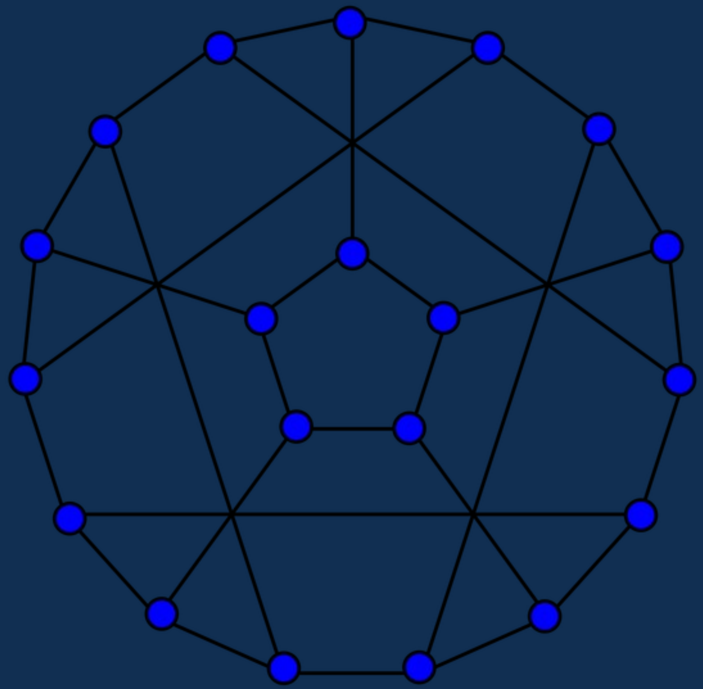
- What are statements of fact in mathematics?
  - Propositions
- How do you formalise an argument?
  - Propositional Logic
- How do you prove a conclusion follows from premises?
  - Equational Reasoning

Groups of things

- How do you represent a collection of things?
  - Sets
- How do collections relate to each other?
  - Set Theory
- How do you work out all possible orders or arrangements of things
  - Combinatorics

Graphs

- How do you represent relationships between things?
  - Graph Theory



Probability

- How do you compare the liklihood of events?
  - Probability
- How do you represent sequences of decisions/events?
  - Decision Trees
- How do you model the range of values a variable can take?
  - Probability Distributions

Statistics

- How do we summarise data and make it understandable?
  - Descriptive Statistics
  - Data Visualisation
- How do we make compelling arguments from data?
  - Inferential Statistics

Back to basics

Basics

- The following is foundational knowledge you’ll need to do this course
  - You don’t necessarily need to know all of this stuff in detail
  - But you need to be familiar with it
- Now is a good chance to make sure you understand it

Equations

- Algebra translates to “the reunion of broken parts”
- We set some stuff equal to some other stuff, e.g.
  - $1 + 2 = 4 - 1$
  - Both sides must remain equal
- Equality is unlike the = button on a calculator which means “calculate”
  - $1 + 2 = 3 + 1 = 4$ is incorrect
  - Be careful laying out equations!
    - $1 + 2 = 3$
    - $3 + 1 = 4$

Equations

Exercise 10:

Which of the following are correct?

1. 100 x 2 = 200 = 10 x 20
2. 55 – 5 = 50 / 2 + 30 = 55
3. a/b = b/a = 1
4. 1 + 2 + 3 + 4 … = 100

Inequalities

- Like equality, inequalities express a relationship between the thing on the left and the thing on the right, e.g.
  - $a < b$, $a > b$
  - Be careful when laying out multiple inequalities
    - $5 < 10 > 15$ is incorrect

Inequalities

Exercise 12:

Let $a = 3, b = 5, c = 8$.

Which of the following are true?

1. $a < b > c$
2. $a < b < c$
3. $a > b < c$
4. $a < b < c$

Variables

- A placeholder for a value that may vary or change, e.g. x in
  - $4 = x + 2$
  - $f(x) = 2x$
- Used to represent, e.g.
  - Numbers
  - Vectors
  - Matrices
  - Functions

Indexes (a_n)

- Indexes are written with subscripts
  - Identify an element of a collection such as a list of numbers
    - A_0 means the 0th element, or element 0,
    - A_1 means the 1st element, or element 1, which is
    - A_n means the nth element, or element n
- Not to be confused with **indices** (also pluralised indexes) that are written in superscript, and show a number has been multiplied by itself

Polynomials

- A polynomial is an expression constants and variables, e.g.
  - $x^2 + 4$
  - $x_3 + x^2 - 2$
- It can be expressed as follows: (a_n is a constant, x is a variable)

$$a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x^1 + a_0$$

- Or as a summation:

$$\sum_{k=0}^{n} a_k x^k$$

Function

- A function is a mapping that associates inputs and outputs, e.g.
  - $f(x) = 2x$, $f(2) = 4$
- Like a function machine. It takes an input and *does something to it*
- The **type** of a function depends on its inputs (domain) and its outputs (range), e.g.
  - $\mathbb{N} \to \mathbb{N}$

Kinds of Function (in $\mathbb{R} \to \mathbb{R}$)

- Polynomial function
  - Has the form of a polynomial, e.g.
  - $f(x) = 2x^2 + 3x + 6$
- Exponential function
  - $f(x) = 2^x$
- Log functions (inverse exponential)
  - $f(x) = x \log 3x + 2$

Operators

- Operators are functions, e.g.
  - a + b = add(a,b)
  - a x b = multiply(a,b)

Exercise 13:

- Give operators that are not:
  - Commutative
  - Associative
  - Distributive over addition

- Addition and multiplication are **commutative**
  - a + b = b + a
  - a x b = b x a
- Addition and multiplication are **associative**
  - a + (b + c) = (a + b) + c
  - a x (b x c) = (a x b) x c
- Multiplication **distributes over addition**
  - a(b + c) = ab + ac

Order of Operations

- Arithmetic operations have **precidence**
  - Different orders may give incorrect results
- Order of Operations (BIDMAS)
  - Brackets
  - Indices
  - Division (note fraction notation)
  - Multiplication
  - Addition
  - Subtraction

Binary Relations

- To foreshadow some stuff in the course...
- Equality and inequalities are **binary relations**
  - They relate two things, *a* and *b*
- Assert a truth claim: *a*R*b* is true

Properties of Binary Relations

- Equality is **reflexive**
  - a = a
- Equlity is **symmetric**
  - If a = b, then b = a
- Equality is **transitive**
  - If a = b, and b = c, then a = c

- Greater than is not **reflexive**
  - a > a is incorrect
- Greater than is not **symmetric**
  - If a > b, then b > a is incorrect
- Greater than is **transitive**
  - If a > b, and b > c, then a > c
- We'll see these again later in the course

Properties of Binary Relations

Exercise 14:

Another binary relation is | “divides”

- 2 | 10 (i.e. 2 divides 10, because 10/2 = 5, a whole number)
- 3 | 9
- 8 | 24

Which of the following are true of the relation | “divides”?

1. Reflexive
2. Symmetric
3. Transitive

Sets

- A set is an **unordered** collection of **elements**, e.g.
  - The set of all pandas
  - The set of all numbers $3 < x < 5$
- Written between curly braces, e.g.
  - $A = \{ a, b, c \}$
  - $B = \{ 1, 2, 3, 4 \}$
  - $C = \{ x \mid x \text{ is a animal} \}$
- See lecture on Set Theory

Summation

- Summation is adding together lots of things
- Often uses Sigma Notation
  - Condense
- See lecture on Sequences and Summation

$$\sum_{k=0}^{n} a_k x^k$$

Summary

Summary

- Why do we need maths in computer science?
  - Computation can be understood mathematically
  - Science is empirical and requires working with data
  - Maths gives us a precice language to communicate knowledge
  - Maths gives us the tools to formally model problems