

Engineering




















KCL Summer School 2019



Puzzle time!

Shape equations!

















The sum of the symbols of each row the square are given.
Can you find the value of each shape?


				16	 =
				14	 =
				16	 =
				20	
17	15	17	17		

Puzzle time!

Shape equations!

The sum of the symbols of each row the square are given.
Can you find the value of each shape?

				16
				14
				16
				20
17	15	17	17	

 = 3

 = 5

 = 4

Puzzle time!

How many of you took this approach?

1. Star = $20/4 = 5$
2. Circle = $(14 - 5)/3 = 3$
3. Triangle = $17 - (2 \times 5) - 3 = 4$

(you could use any number of the remaining rows or columns for the third step)



1. Look for a row or column containing only one type of symbol
2. Solve for that symbol
3. Look for a row or column containing that symbol and only one other symbol
4. Solve for the other symbol by replacement
5. Solve for the third symbol using any remaining row or column

Shape equations!

The sum of the symbols of each row the square are given.
Can you find the value of each shape?

	●	▲	▲	★	16
2.	★	●	●	●	14
	▲	●	★	▲	16
1.	★	★	★	★	20
	17	15	17	17	

3.

● = 3
★ = 5
▲ = 4

Puzzle time!

If you did, then, probably without realising it, you were applying the principles of **Computational Thinking**

Shape equations!

The sum of the symbols of each row the square are given.
Can you find the value of each shape?

2.

●	▲	▲	★	16
★	●	●	●	14
▲	●	★	▲	16

1.

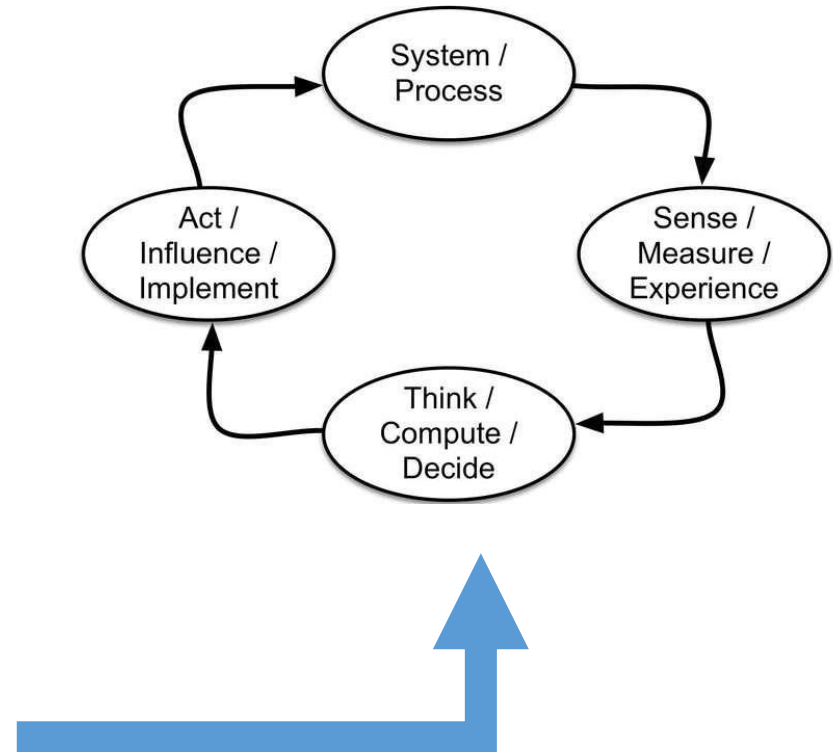
★	★	★	★	20
17	15	17	17	

3.

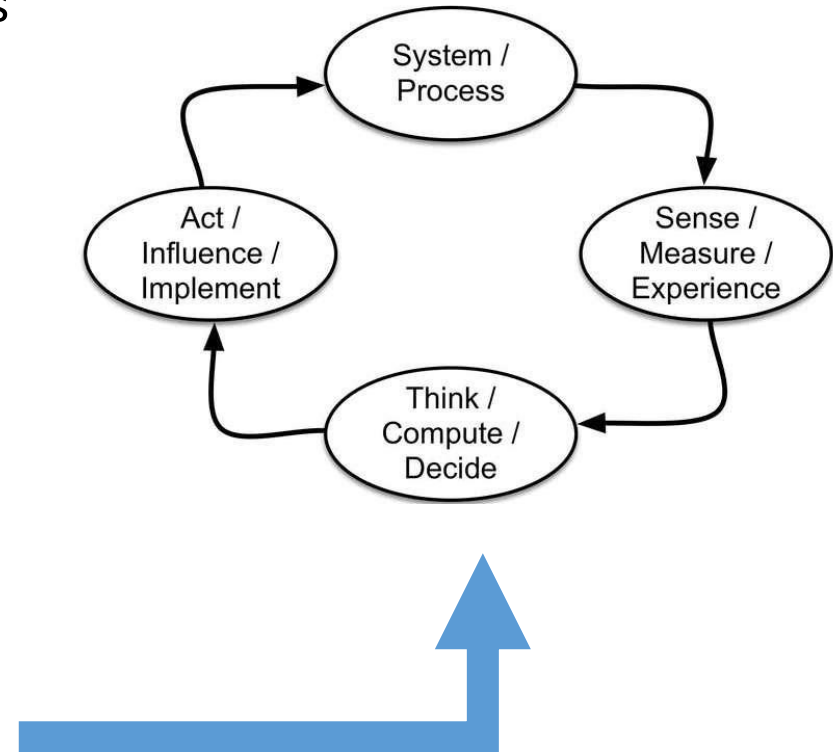
● = 3
★ = 5
▲ = 4

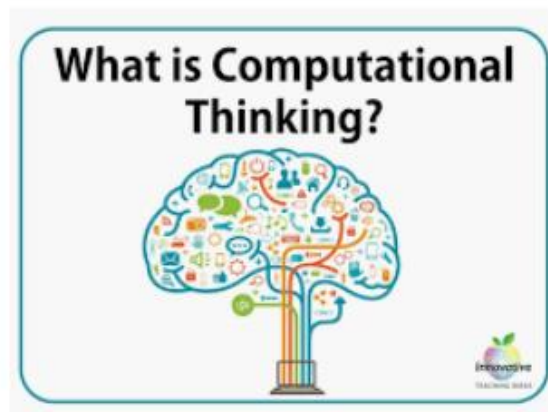
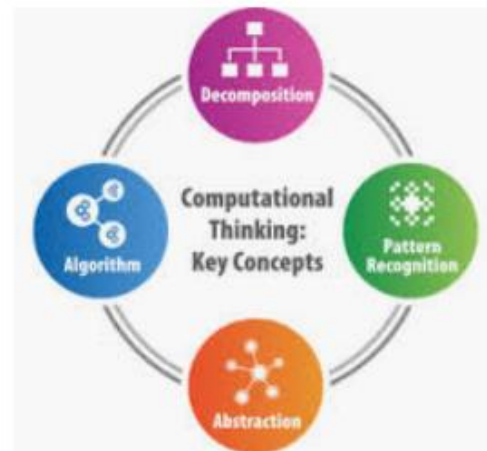
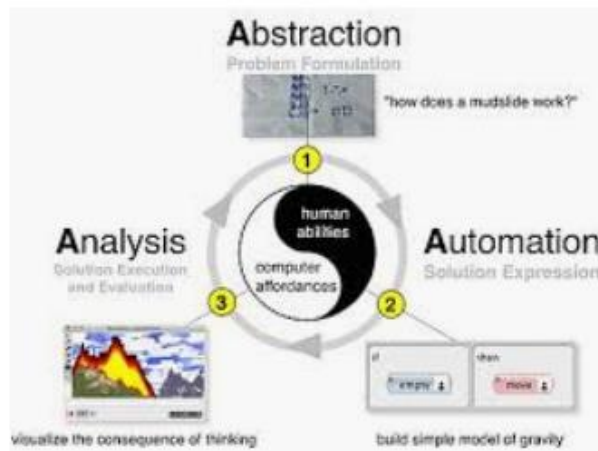
Computational Thinking

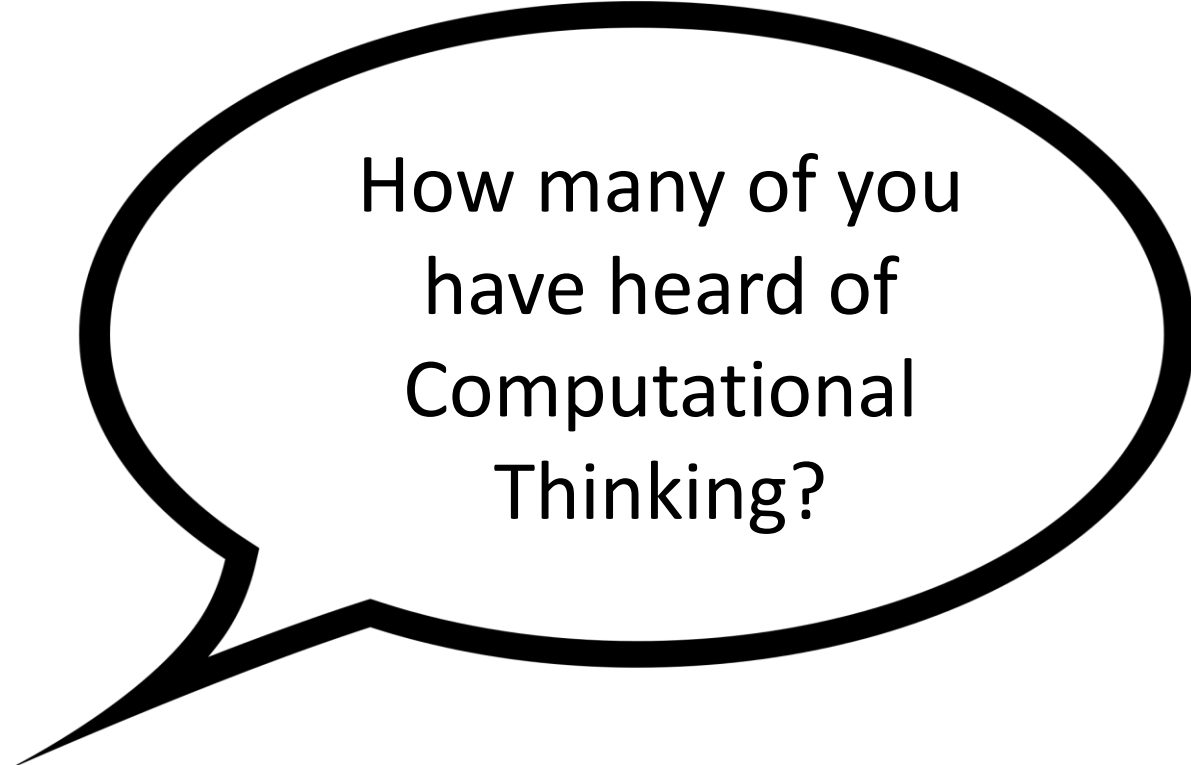
We've talked about Engineering being there to help us solve problems



Now I want to introduce an increasingly-prominent general approach to problem-solving that is well-suited to the Engineering Domain







How many of you
have heard of
Computational
Thinking?

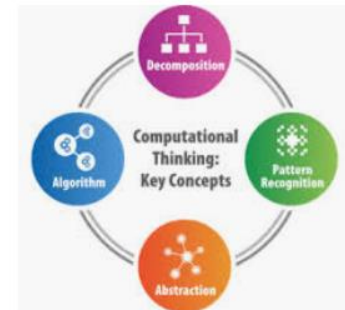
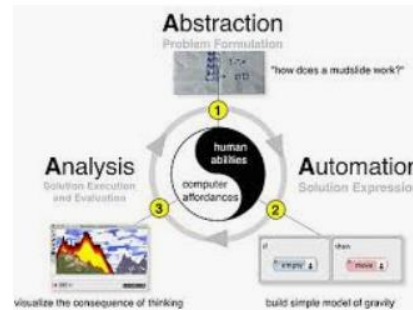


defines “Computational Thinking” as:

Computational thinking (CT) is a problem-solving process that includes (but is not limited to) the following characteristics:

- Formulating problems in a way that enables us to use a computer and other tools to help solve them.
- Logically organizing and analyzing data
- Representing data through abstractions such as models and simulations
- Automating solutions through algorithmic thinking (a series of ordered steps)
- Identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources
- Generalizing and transferring this problem solving process to a wide variety of problems

But, it is really important to understand that Computational Thinking is not “thinking like a computer”



If anything, it is much
closer to thinking like a
chef



To understand what I mean
by that, we're going to
explore the different
elements of computational
thinking



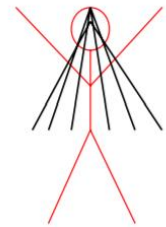
But, first, a bit of background.

CT has really come to the fore thanks in large part to the advocacy of Prof. Jeannette Wing, formerly of Carnegie Mellon University.



Computational Thinking

Jeannette M. Wing



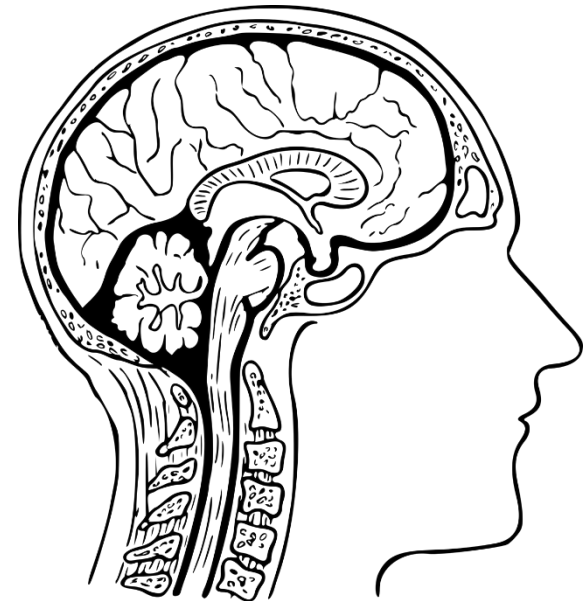
President's Professor of Computer Science and Department Head
Computer Science Department
Carnegie Mellon University

© 2007 Jeannette M. Wing

Prof. Wing is a Computer Scientist, and she saw in **the approach** that she and her colleagues had evolved to tackling problem-solving in that field, something that could be applied to most every type of problem-solving.

Computational Thinking: What It Is and Is Not

- Conceptualizing, not programming
 - Computer science is not just computer programming
- Fundamental, not rote skill
 - A skill every human being needs to know to function in modern society
 - Rote: mechanical. Need to solve the AI Grand Challenge of making computers “think” like humans. Save that for the second half of this century!
- A way that humans, not computers think
 - Humans are clever and creative
 - Computers are dull and boring

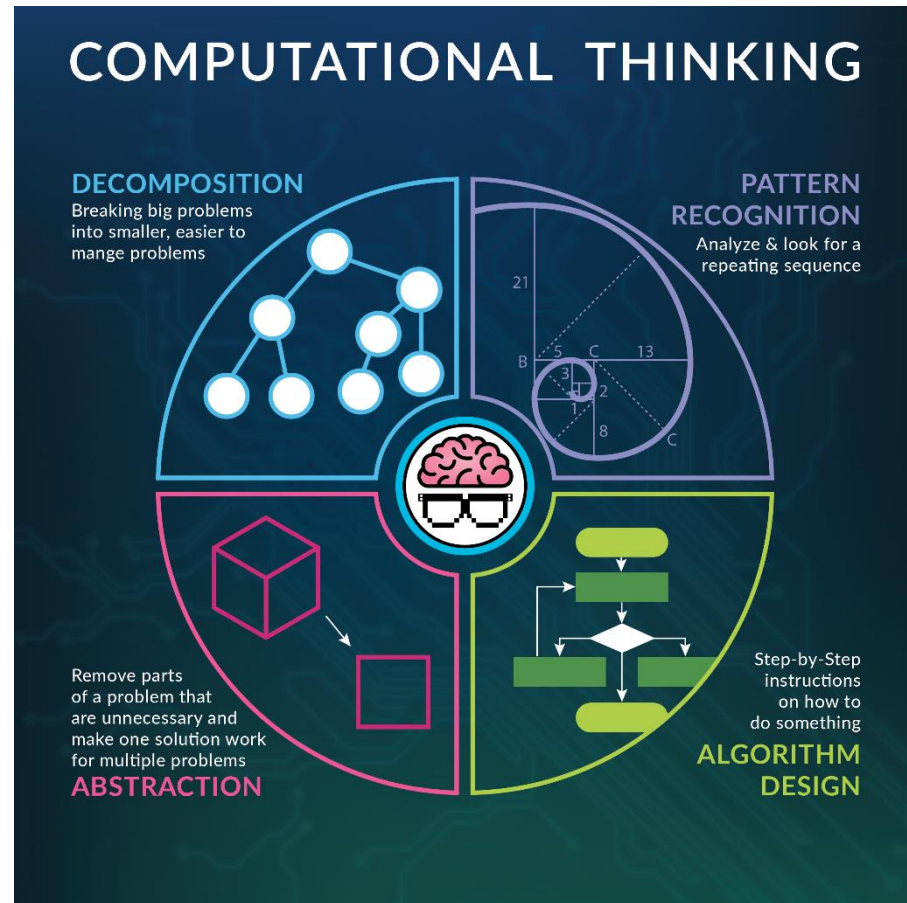


Prof. Wing is a Computer Scientist, and she saw in the approach that she and her colleagues had evolved to tackling problem-solving in that field, something that could be applied to most every type of problem-solving.



So, let's look at the elements of CT:

1. Decomposition
2. Pattern Recognition
3. Abstraction
4. Algorithm Design (AKA "Automation")



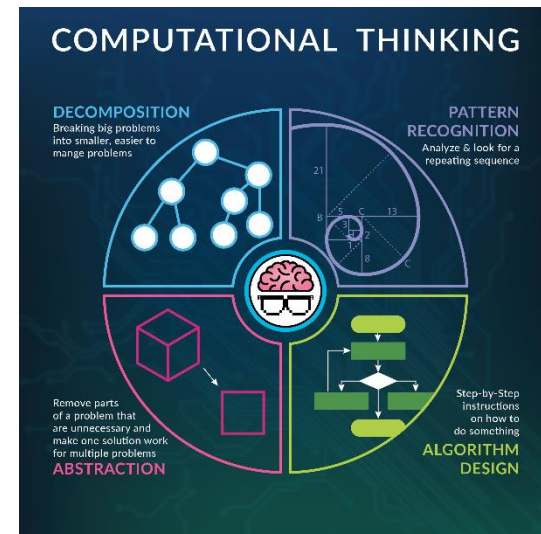
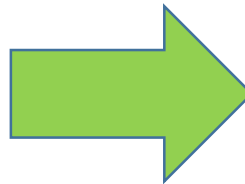
Let's breakdown our approach to solving our little puzzle into its essential elements and see how they correspond to the elements of CT

Shape equations!

The sum of the symbols of each row the square are given.
Can you find the value of each shape?

●	▲	▲	★	16
★	●	●	●	14
▲	●	★	▲	16
★	★	★	★	20
17	15	17	17	

$$\begin{aligned} \bullet &= 3 \\ \star &= 5 \\ \blacktriangle &= 4 \end{aligned}$$



Step-by-step

Decomposition: break the problem down into smaller, more manageable parts

- Solve for one symbol at a time

Shape equations!

The sum of the symbols of each row the square are given.
Can you find the value of each shape?

●	▲	▲	★	16	● = 3
★	●	●	●	14	★ = 5
▲	●	★	▲	16	▲ = 4
★	★	★	★	20	
17	15	17	17		




















Step-by-step

Pattern Recognition: look for repeating sequence(s)

- Four stars in one row
- Three circles [and one star] in another row
- Etc.

Shape equations!

The sum of the symbols of each row the square are given.
Can you find the value of each shape?

				16	 = 3  = 5  = 4
				14	
				16	
				20	
17	15	17	17		

Step-by-step

Abstraction: ignore elements of the problem that aren't needed for the [general] solution

- We don't need to think about these elements to solve the problem

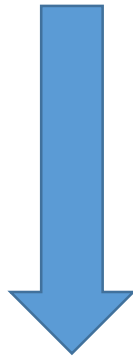
Shape equations!

The sum of the symbols of each row the square are given.
Can you find the value of each shape?

●				16	● = 3
☆	●	●	●	14	☆ = 5
▲				16	▲ = 4
☆	☆	☆	☆	20	
17	15	17	17		

Step-by-step

**Algorithmic Design (AKA
“Automation”):** step by step
instructions



1. Look for a row or column containing only one type of symbol
2. Solve for that symbol
3. Look for a row or column containing that symbol and only one other symbol
4. Solve for the other symbol by replacement
5. Solve for the third symbol using any remaining row or column

Shape equations!

The sum of the symbols of each row the square are given.
Can you find the value of each shape?

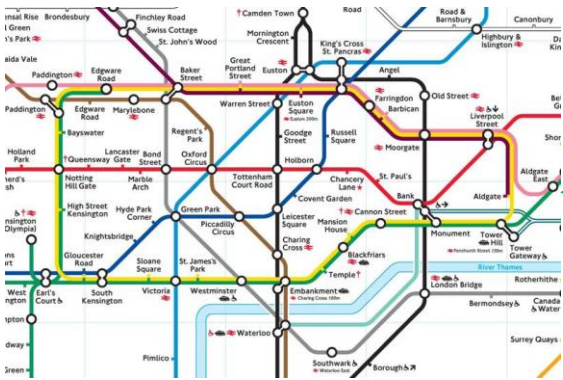
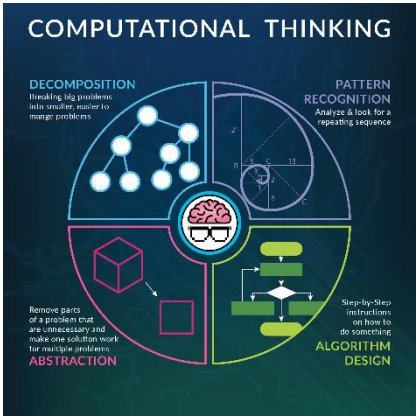
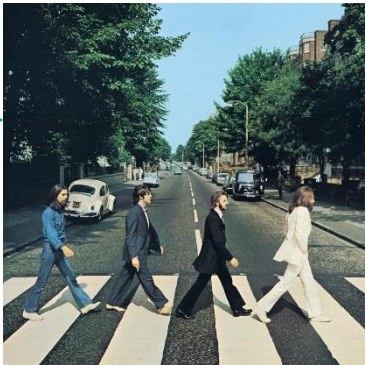
	●	▲	▲	★	16
2.	★	●	●	●	14
	▲	●	★	▲	16
1.	★	★	★	★	20
	17	15	17	17	
3.					

● = 3

★ = 5

▲ = 4

Of course, examples of these concepts turn up everywhere in our daily lives



And, all of them can be applied to the process of cooking a meal at one stage or another

decomposition – break the meal down into courses, break down the order into components, etc.;

pattern recognition – different foods prepared and/or cooked the same way (bake a lasagne, bake a cake...);

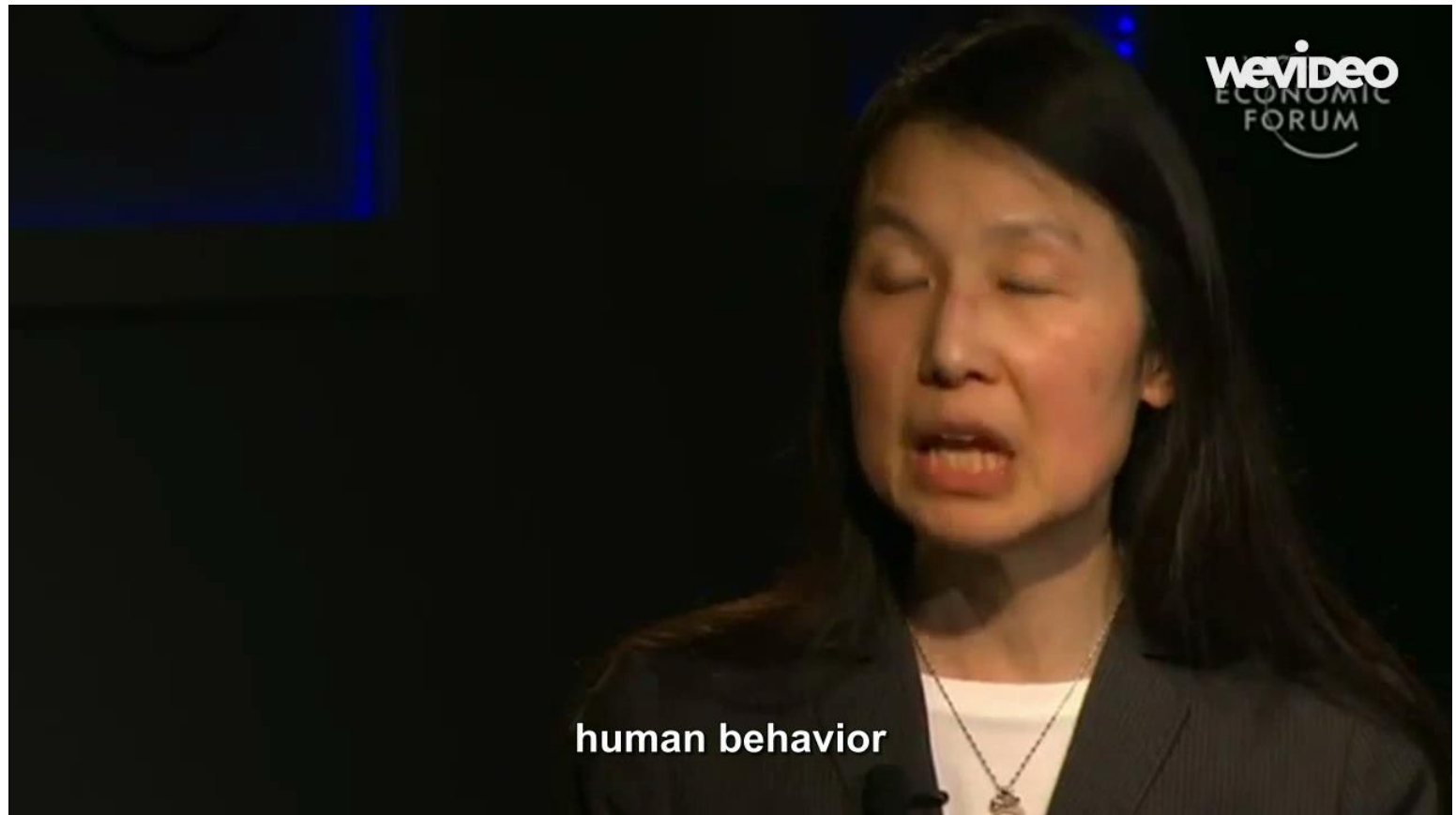
abstraction – don't need to know how a cooker keeps a constant temperature, just that it does;

algorithmic design – the recipe, of course

(Computational Thinking is thinking like a chef...)



(And, I'm not the only person
who thinks this 😊)



Algorithm Design: Problem Strategies

We've discussed 3 of the general skills involved in computational thinking, *decomposition, pattern recognition, abstraction*

Now let's *briefly* talk about the 4th skill, **algorithm design**, which lets us break-down solutions to problems into discrete steps that can be repeated.

When learning about algorithm design, it is useful to discuss specific *types* of algorithm design *strategies*.

Real world problems can be *large and complex*, often not matching exactly one type of problem, but forming a combination of types.

Algorithmic puzzles are good ways of learning to recognise these problem types, and for learning approaches to solving them *using* decomposition, pattern recognition, and abstraction.

Algorithm Design Strategies (A non-exhaustive list!)

Exhaustive Search

Brute force a solution – try all possible candidate solutions.

Backtracking

Build solutions one step at a time. If no further steps possible, rewind and try a different action (tree-structure).

Decrease and Conquer

Find a way to reduce candidate answers, until the true solution becomes obvious.

Divide and Conquer

Split the main problem into simpler subproblems that can be solved and combined to give the main solution.

Transform and Conquer

Transform the problem into another problem that is easier to solve – i.e. find a good representation.

Greedy Approach

Solves a problem in a sequence of steps, with each step seeking to maximise the immediate reward.

Iterative Improvement

Start with a rough guess at the solution, which is easier to obtain. Iteratively improve the guess

Dynamic Programming*

Similar to divide and conquer, but subproblem solutions are stored to let you solve many problems.

Analysis Techniques

Algorithmic/mathematical analysis of a problem to find closed-form solutions.

Invariants

Identifying properties of the problem that are constant. Often used to show a problem is unsolvable.

Algorithm Design Strategies (A non-exhaustive list!)

Exhaustive Search

Brute force a solution – try all possible candidate solutions.

Backtracking

Build solutions one step at a time. If no further steps possible, rewind and try a different action (tree-structure).

Decrease and Conquer

Find a way to reduce candidate answers, until the true solution becomes obvious.

Divide and Conquer

Split the main problem into simpler subproblems that can be solved and combined to give the main solution.

Transform and Conquer

Transform the problem into another problem that is easier to solve – i.e. find a good representation.

Greedy Approach

Solves a problem in a sequence of steps, with each step seeking to maximise the immediate reward.

Iterative Improvement

Start with a rough guess at the solution, which is easier to obtain. Iteratively improve the guess

Dynamic Programming*

Similar to divide and conquer, but subproblem solutions are stored to let you solve many problems.

Analysis Techniques

Algorithmic/mathematical analysis of a problem to find closed-form solutions.

Invariants

Identifying properties of the problem that are constant. Often used to show a problem is unsolvable.

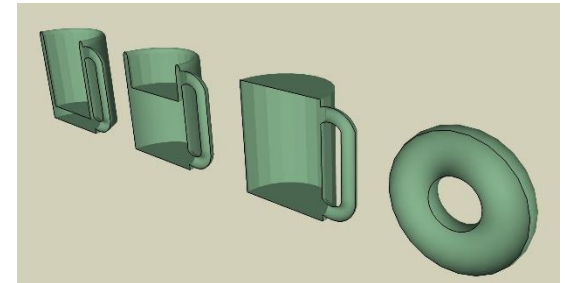
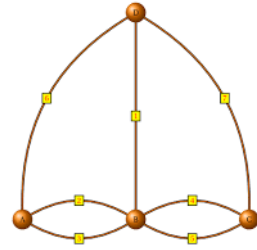
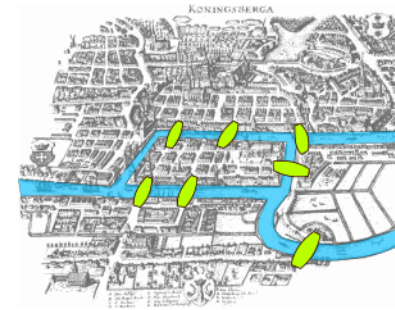
Let's focus on these for now

Invariants

Identifying properties of the problem that are constant. Often used to show a problem is unsolvable.

Seven Bridges of Königsberg

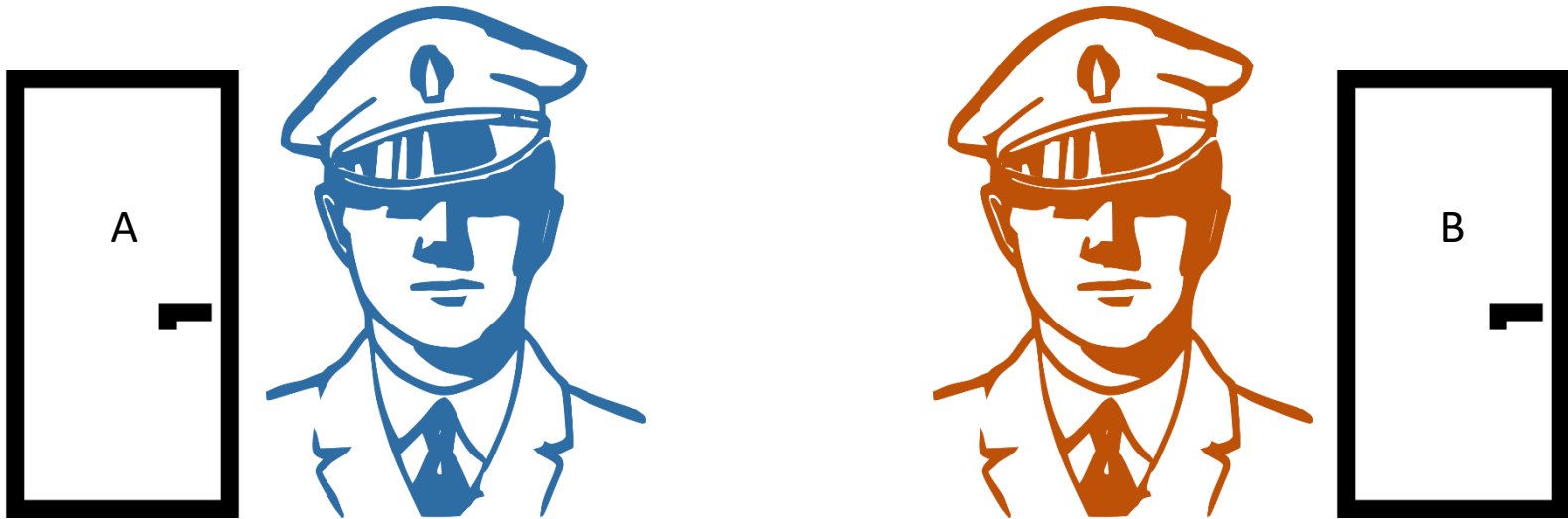
- Challenge was to design a tour of the city, which started and ended in the same location, and crossed each of the 7 city bridges exactly once.
- Euler ("Oil-er") transformed the city map into a *graph* representation, keeping only what was important with landmasses forming *nodes* and bridges forming *edges*
 - *Topological transformation.*
- Euler identified the pattern that during a walk, *the number of times one enters a non-terminal vertex equals the number of times one leaves it* (entering and then leaving the land mass).
- If every bridge has been traversed exactly once, it follows that, for each land mass (except for the ones chosen for the start and finish), the number of bridges touching that land mass **must be even** (half of them, in the particular traversal, will be traversed "toward" the landmass; the other half, "away" from it).
- **However**, all four of the land masses in the original problem are touched by an odd number of bridges.
- This leads to a **contradiction**. The tour is not possible!
- Invariants often, but not always, used to prove something is not possible.
- Euler shows that the possibility of a walk through a graph, traversing each edge exactly once, depends on the degrees of the nodes. The degree of a node is the number of edges touching it.
- Such a walk is now called an *Eulerian path* or *Euler walk* in his honor.



https://en.wikipedia.org/wiki/Seven_Bridges_of_K%C3%B6nigsberg

Invariants

Identifying properties of the problem that are constant. Often used to show a problem is unsolvable.



Two doors. One door is your escape route, one is a trap.

Two guards, one always lies, one always tells the truth. You don't know which one is the liar.

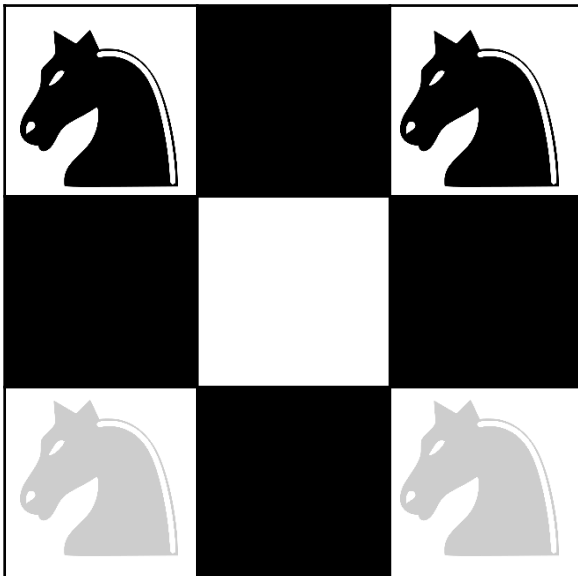
How do you find out which door is the correct one?

Algorithm Design Strategies: Examples

Transform and Conquer

Transform the problem into another problem that is easier to solve – i.e. find a good representation.

Guarini's Puzzle



Can the white knights positions be swapped with the black knights positions?

If so, in how many moves?

What is the algorithm?

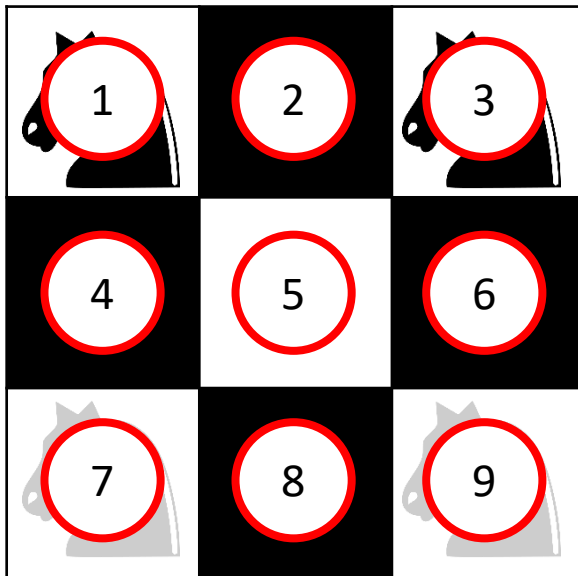
Can you prove your solution holds?

(To Print)

Transform and Conquer

Transform the problem into another problem that is easier to solve – i.e. find a good representation.

Guarini's Puzzle



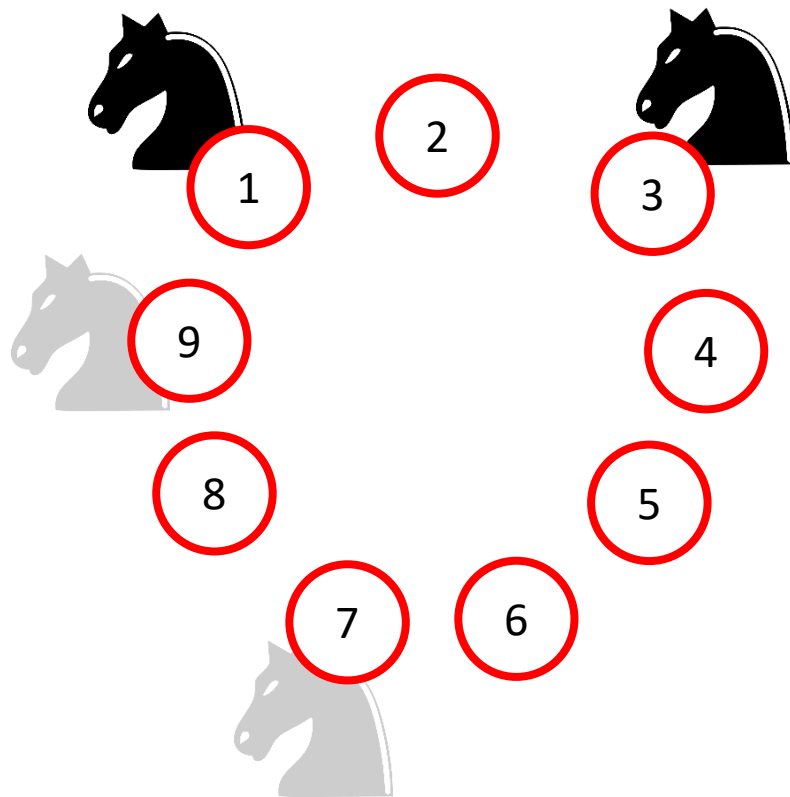
Clue 1:

- Label each position with a number.
- From position 1, which positions could a knight reach? What about from position 2? Etc. How might all these possibilities be linked?

Transform and Conquer

Transform the problem into another problem that is easier to solve – i.e. find a good representation.

Guarini's Puzzle



Clue 2:

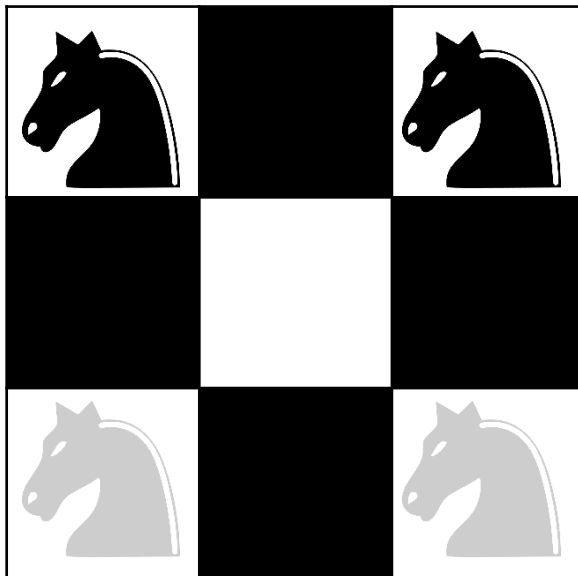
- Draw the positions as *nodes* in a circle.
- What is the relationship between these nodes?
- What does this tell us about the path the knights can take?

Algorithm Design Strategies: Examples

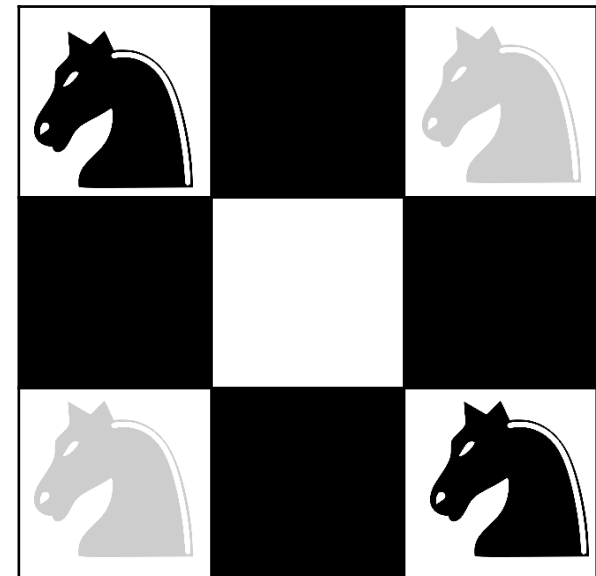
Transform and Conquer

Transform the problem into another problem that is easier to solve – i.e. find a good representation.

Guarini's Puzzle 2



How?

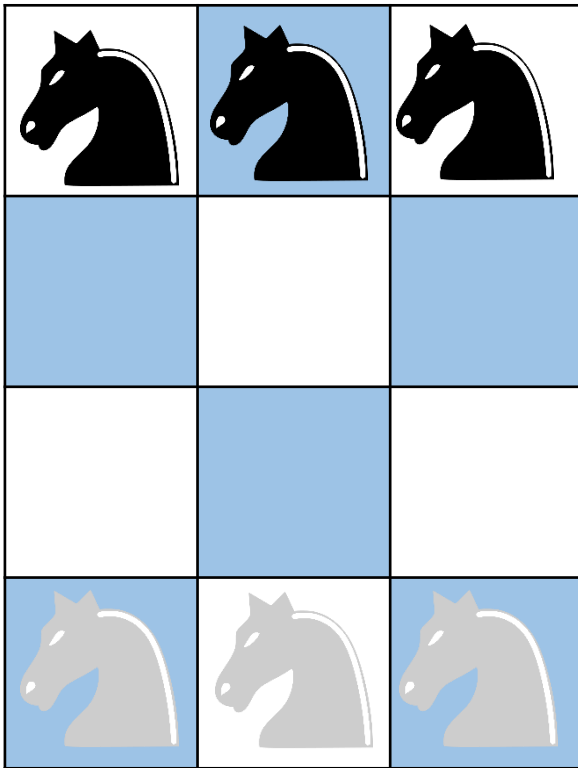


Algorithm Design Strategies: Challenge!

Transform and Conquer

Transform the problem into another problem that is easier to solve – i.e. find a good representation.

Guarini's Puzzle 3



Can the positions of the black knights be swapped with the white knights? If possible, what is the minimum number of moves for this? Show proof.

(To Print)

Carl Friedrich Gauss. *Hugely* influential 18th/19th century mathematician and physicist, >100 things named after him.

Famous story from when he was 7 years old in class.

The teacher wanted some quiet time, so told the students to sum all the numbers from 1 to 100. Gauss, being Gauss (and applying CT of course), was finished within a few moments.

The teacher was annoyed, thinking he was not doing the task, but found Gauss had reached the correct answer.

What did he do?

He found a *pattern*.

He noted that adding the first and last numbers in the list, and moving inward, gives the same sum.

$$1 + 100 = 101, 2 + 99 = 101, 3 + 98 = 101, \dots, 50 + 51 = 101$$

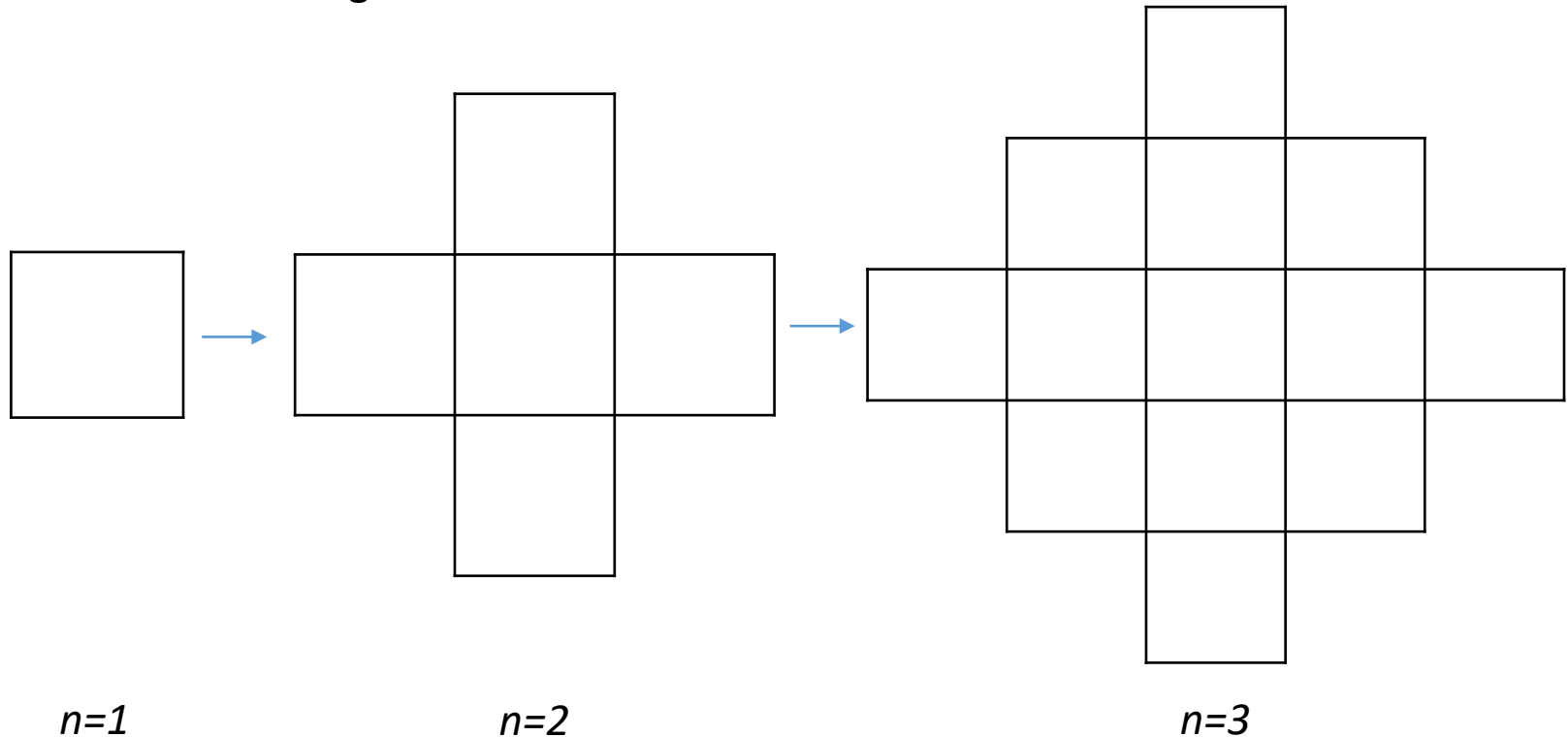
There will be 50 of these sum terms (half the length of the list), therefore the sum of the list will be

$$50 * 101 = 5050.$$

The *algorithm* for the sum of a list of numbers from 1 to n (where $n = 100$ in this example) is therefore

$$\frac{n(n + 1)}{2}$$

Challenge!



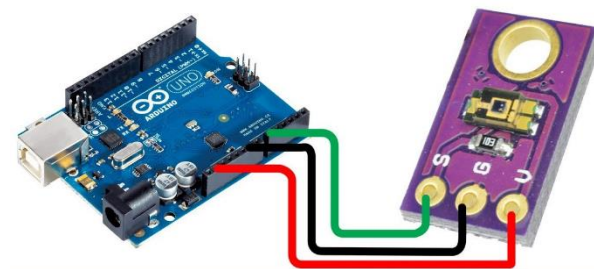
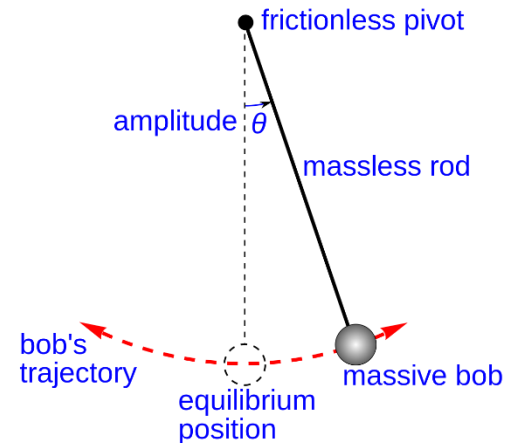
Given an initial 2D unit square, if we add unit squares to each of the initial square's faces at each time step, *how many unit squares will there be in the resulting shape after n time steps?*

(To Print)

Tomorrow, in the lab, you are going to be using a light sensor (a sensor that measures the amount of light falling on its surface) to measure the period of a pendulum

How?

Come up with a recipe for doing this (note: you don't need to know anything about how the light sensor or how the microcontroller it's attached to works to do this)



(To Print)

(you'll receive full instructions for this tomorrow – this is just a little something extra for you to try)

Summary

- Introduced Computational Thinking
- Solved Some Puzzles
- Thought about tomorrow

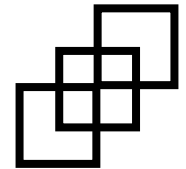
Further Reading

- Google online courseware for Computational Thinking
<https://edu.google.com/resources/programs/exploring-computational-thinking/#!ct-materials>
- Algorithmic Puzzles, A. Levitin, M. Levitin, 2011 (very accessible book on the topic).
- Project Euler <https://projecteuler.net/> Large collection of CT/algorithmic challenges for those wanting to improve their maths and programming skills.

Quick-fire puzzles!



1. A King has 8 barrels of water on his ship. He knows one is poisoned, and will surely kill, but not which one. How many loyal servants must taste the water to find the poisoned barrel?
2. You have 8 gold coins, and you are sure there is one slightly lighter fake in the pile. You have a balance scale, but no reference weights. What is the minimum number of comparisons required to find the fake coin?
3. You have 5 pairs of blue gloves, 3 pairs of red gloves, and 2 pairs of yellow gloves in your drawer. It is dark and you cannot see. How many individual gloves must you take out to *guarantee* you have a matching pair of any colour? How many must you take out to guarantee you have a matching pair of each colours?
4. Four people must cross a bridge at night, but have only one flashlight. Only one person can cross at a time, and one of them must have the flashlight. Each person completes the walk in different times. Person A can do it in 1 minutes, B in 2 minutes, C in 5 minutes, and D in 10 minutes. What is the shortest amount of time required to get everyone across?
5. Can you draw the figure shown, without lifting your pen off the page and without crossing over any lines? Prove yes or no.
6. You have two crystal balls, and want to find out at what height they will break from when dropped from windows in a tall building (100 floors). You can drop a ball as many times as you want until it breaks (it breaks due to being dropped from too high). What is the minimum number of drops required to find the guaranteed highest safe floor to drop a crystal ball from?



(To Print)