CS1010 http://www.comp.nus.edu.sg/~cs1010/

Programming Methodology

UNIT 1

Computational Thinking

Unit 1: Computational Thinking

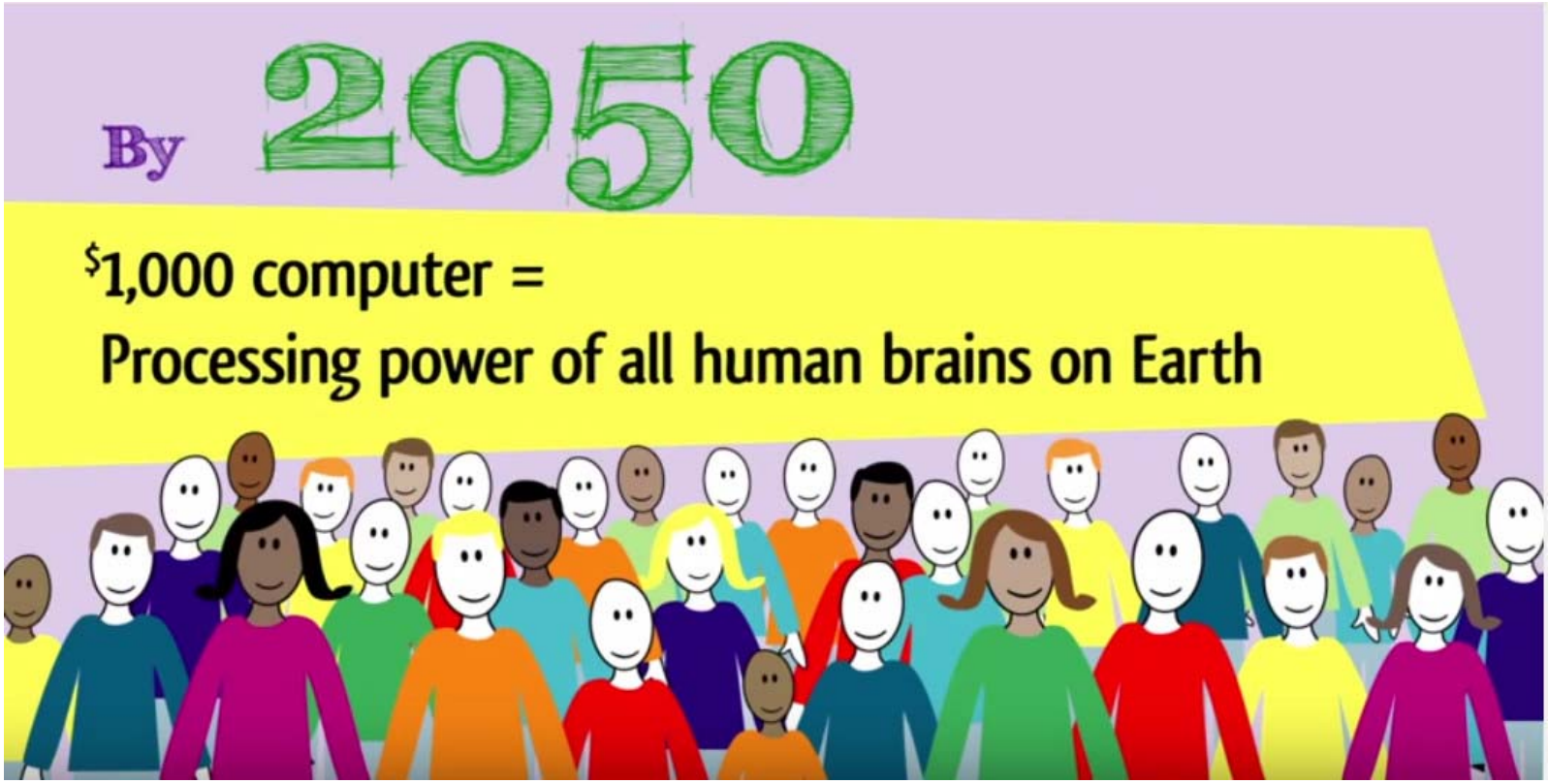
1. Computational Thinking
2. Polya's Problem Solving Process
3. Understanding the Problem
4. Problem Formulation/Definition

Acknowledgement:

We would like to thank A/P Leong Hon Wai and A/P Leow Wee Kheng for allowing us to use part of the materials from their module GET1031 Computational Thinking for this unit.

A digital age skill for everyone

A four-minute video from ISTE:
https://www.youtube.com/watch?v=VFcUgSYyRPg



Viewpoint | Jeannette M. Wing

Communications of the ACM
March 2006/Vol. 49, No. 3

Computational Thinking

It represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use.

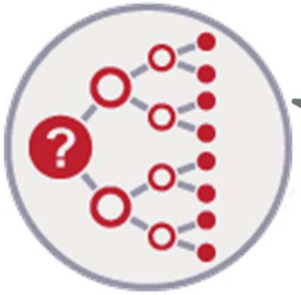
Computational thinking builds on the power and limits of computing processes, whether they are executed by a human or by a machine. Computational

cisely. Stating the difficulty of a problem accounts for the underlying power of the machine—the computing device that will run the solution. We must consider the machine's instruction set, its resource constraints, and its operating environment.

In solving a problem efficiently, we might further

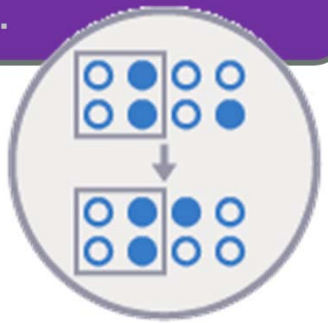
Computational Thinking

Decomposition



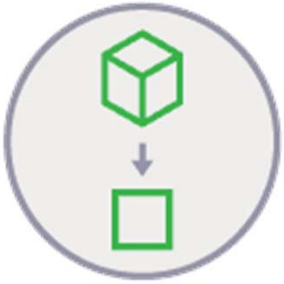
Breaking the problem into smaller, more manageable parts.

Recognising which parts are the same and the attributes that define them.



Pattern recognition

Abstraction

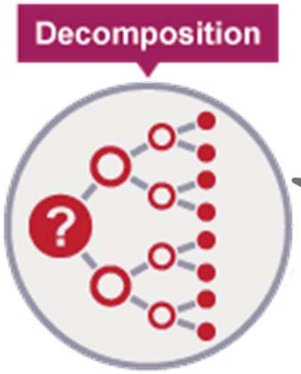


Filtering out info not needed and generalising info that is needed.

Creating solutions using a series of ordered steps.

Algorithms

CT: Decomposition



Breaking the problem into smaller, more manageable parts.

Why is this important?

- To manage and overcome complexity.
- Leads to pattern recognition and generalization.

Example:

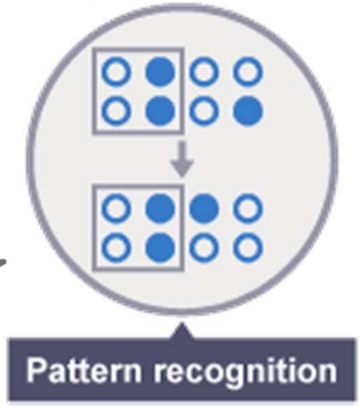
- Baking a cake

Video:

https://www.youtube.com/watch?v=yQVTijX437c

CT: Pattern Recognition

Recognising which parts are the same and the attributes that define them.



Why is this important?

- Finding similarities and patterns in order to solve complex problems more efficiently.

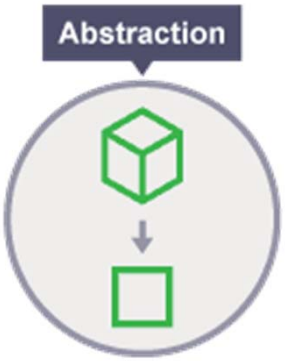
Example:

- Detecting and classifying whether an email is a spam

Video:

https://www.youtube.com/watch?v=SixLnIDV1yY

CT: Abstraction



Filtering out info not needed and generalising info that is needed.

Why is this important?

- A means to distil what is essential.
- To create models, rules, principles or theories to test predicted outcomes.
- So that one solution works for multiple problems with similar nature.

Examples:

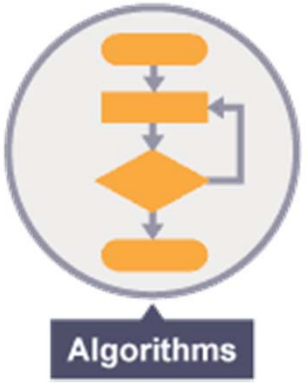
- Google maps
- Baking a cake

Video:

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

CT: Algorithms

Creating solutions using a series of ordered steps.



Why is this important?

- Algorithms are plans/procedures that can be communicated at an abstract level and implemented on different platforms.
- Can be analysed and studied.

Example:

- A recipe to bake chocolate cake.

Video:

https://www.youtube.com/watch?v=N91oCQbWUvA

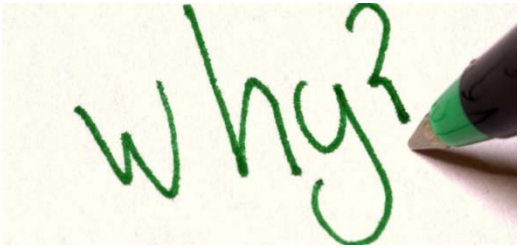
Computational Thinking (CT) is NOT restricted to Computer Science!

Computational Thinking Concept	Subject Area Application
Break a problem into parts or steps	**Literature:** Break down the analysis of a poem into analysis of meter, rhyme, imagery, structure, tone, diction, and meaning.
Recognize and find patterns or trends	**Economics:** Find cycle patterns in the rise and drop of the country's economy.
Develop instructions to solve a problem or steps for a task	**Culinary Arts:** Write a recipe for others to use.
Generalize patterns and trends into rules, principles, or insights	**Mathematics:** Figure out the rules for factoring 2nd-order polynomials
	Chemistry: Determine the rules for chemical bonding and interactions.

Today, you learned about Computational Thinking (CT).

But… today is NOT the only lesson on CT.

The concepts of CT will be reinforced throughout the semester.



Computational Thinking

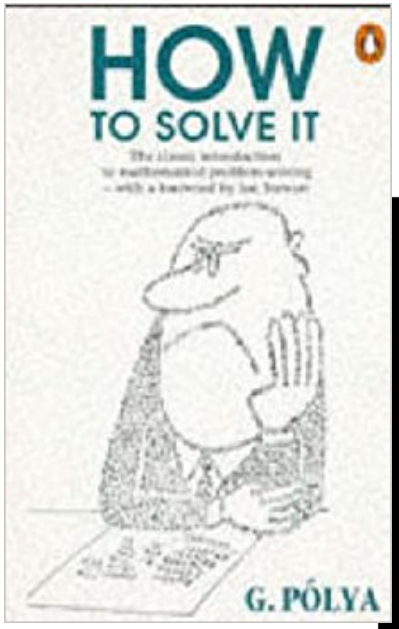
It represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use.

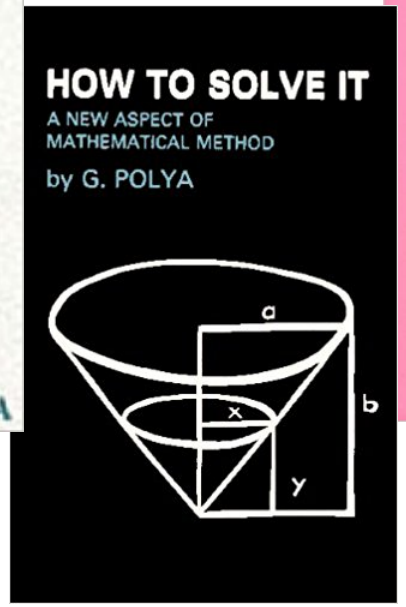
Computational Thinking

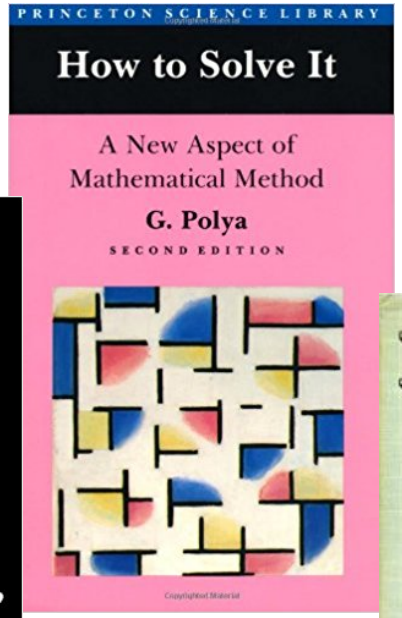
Problem Solving

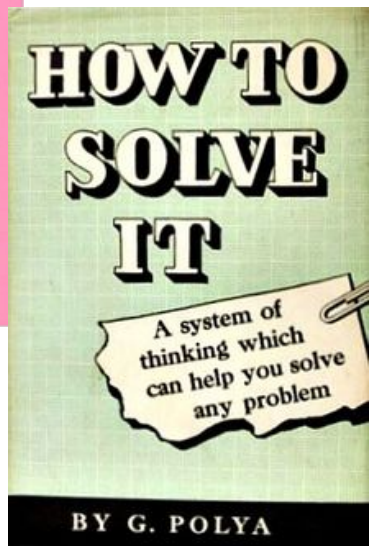
The most widely cited reference for problem solving in all disciplines!

George Polya
(1887 – 1985)









A few Polya's Quotes

"A great discovery solves a great problem, but there is a grain of discovery in the solution of any problem. Your problem may be modest, but if it challenges your curiosity and brings into play your inventive faculties, and if you solve it by your own means, you may experience the tension and enjoy the triumph of discovery."

"Quite often, when an idea that could be helpful presents itself, we do not appreciate it, for it is so inconspicuous. The expert has, perhaps, no more ideas than the inexperienced, but appreciates more what he has and uses it better."

"If you can't solve a problem, then there is an easier problem you can solve: find it."

Steps to Problem Solving

1. Understand the Problem

Can you state the problem in your own words?

What are you trying to find or do?

What are the unknowns?

What info do you obtain from the problem?

What info, if any, is missing or not needed?

2. Make a Plan

Look for a pattern.

Remember related problems.

Break problem down into different parts.

Make a table, diagram or write equation.

Use guess and check.

Work backward.

Identify a subgoal.

3. Do the Plan

Implement the strategy in step 2.

Check each step of the plan as you do it.

Keep an accurate record of your work.

Organize your work into easy to understand visuals.

Double check your math work.

4. Look Back

Check the results in the original problem.

Interpret the solution in terms of the original problem. Does your answer make sense? Is it reasonable?

Determine whether there is another method of finding the solution.

If possible, determine other related or more general problems for which the techniques will work.

Develop habit of ASKING

1. Understand the Problem

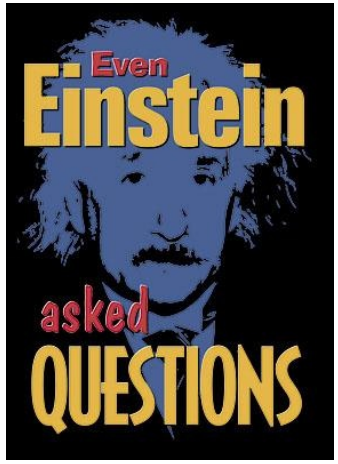
Can you state the problem in your own words?

What are you trying to find or do?

What are the unknowns?

What info do you obtain from the problem?

What info, if any, is missing or not needed?



Do not just understand the solution;
Understand ***how*** and ***why*** of getting the solution.

Do I understand the problem?

Why does the solution work?

How can I invent such a solution?

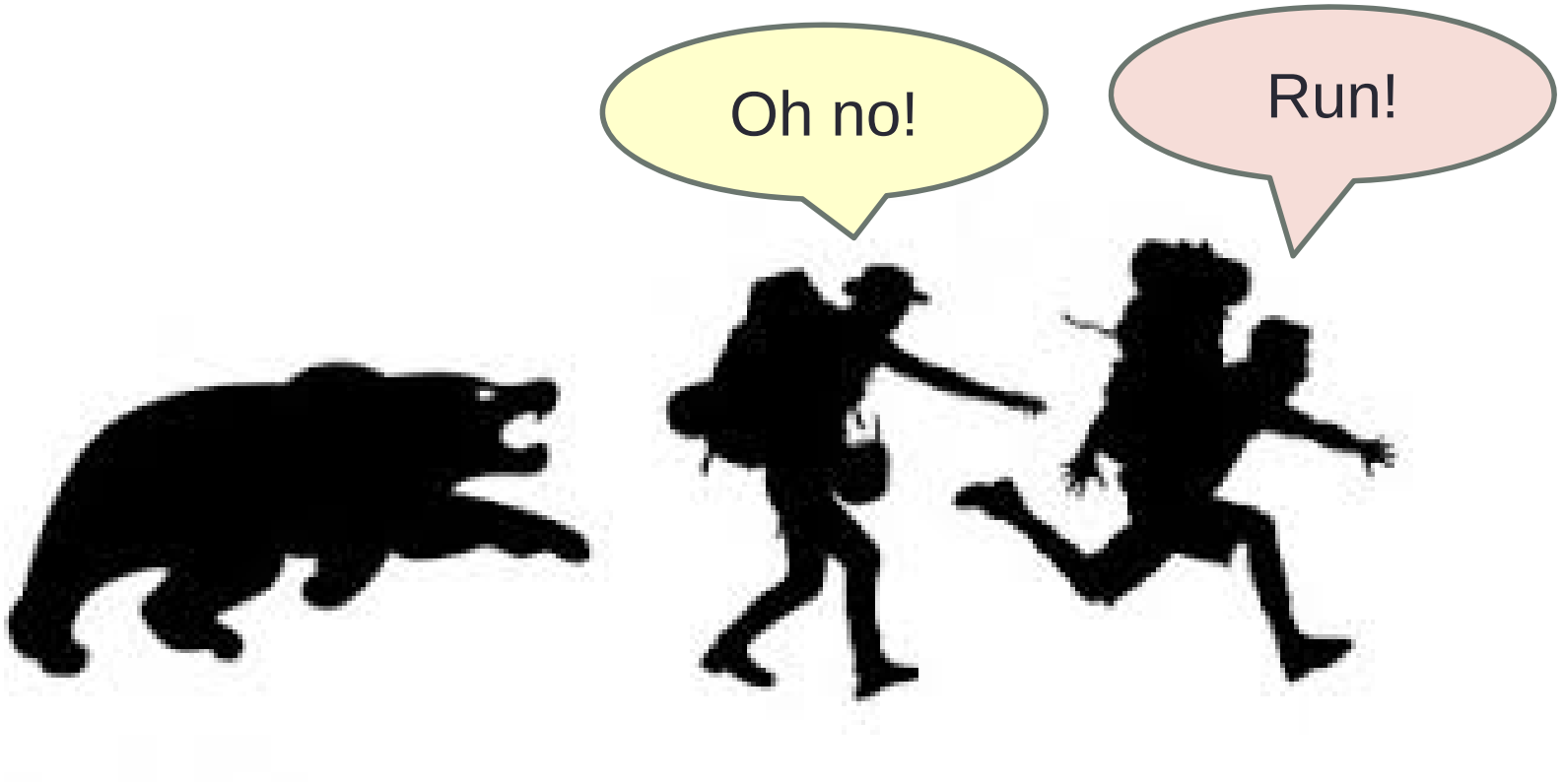
Do I understand the solutions?

Why did he do it like that?

Can I do it differently?

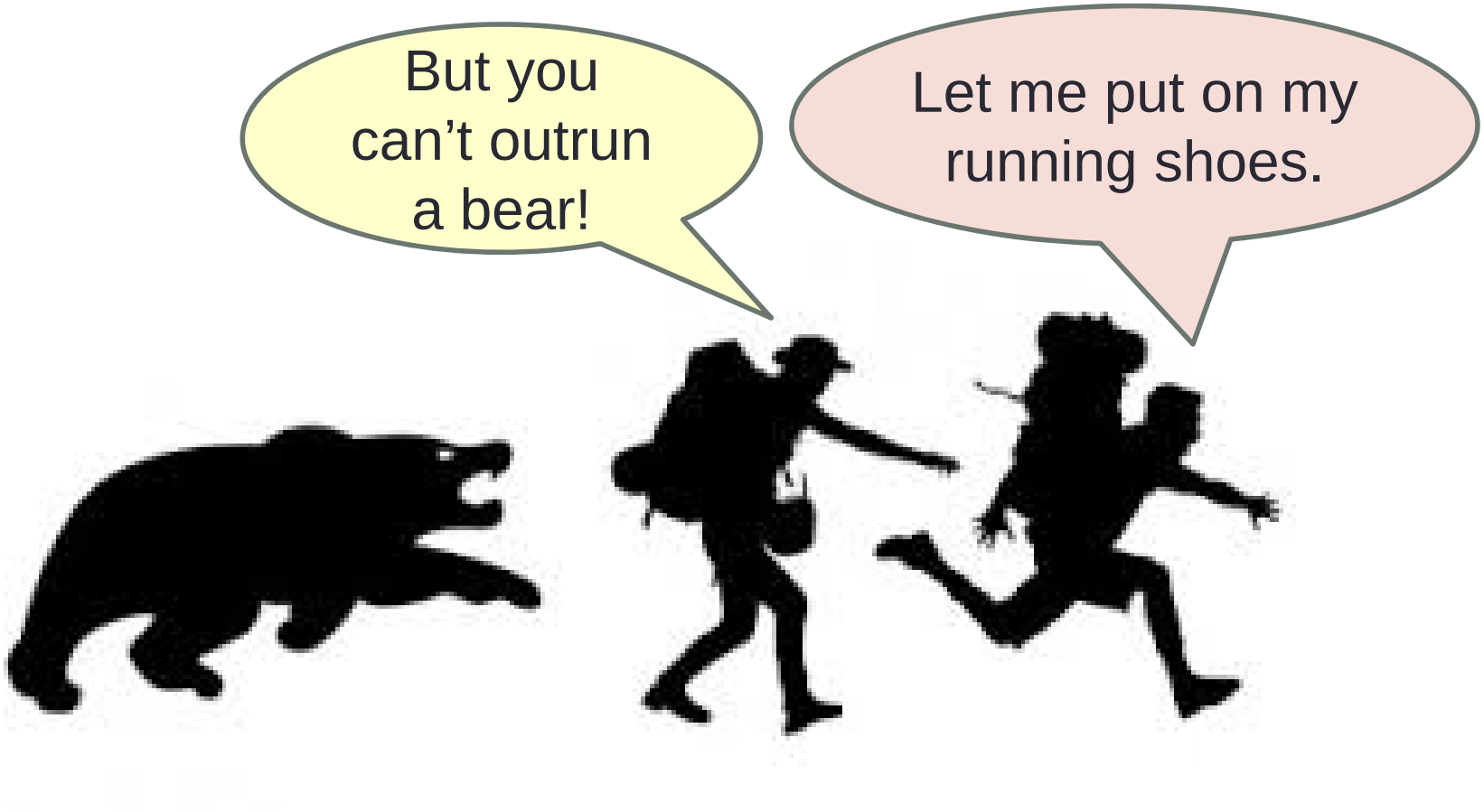
How *hard* can "understanding the problem" be?

(1) The Hungry Bear Problem



How ***hard*** can "understanding the problem" be?

(1) The Hungry Bear Problem



How *hard* can "understanding the problem" be?

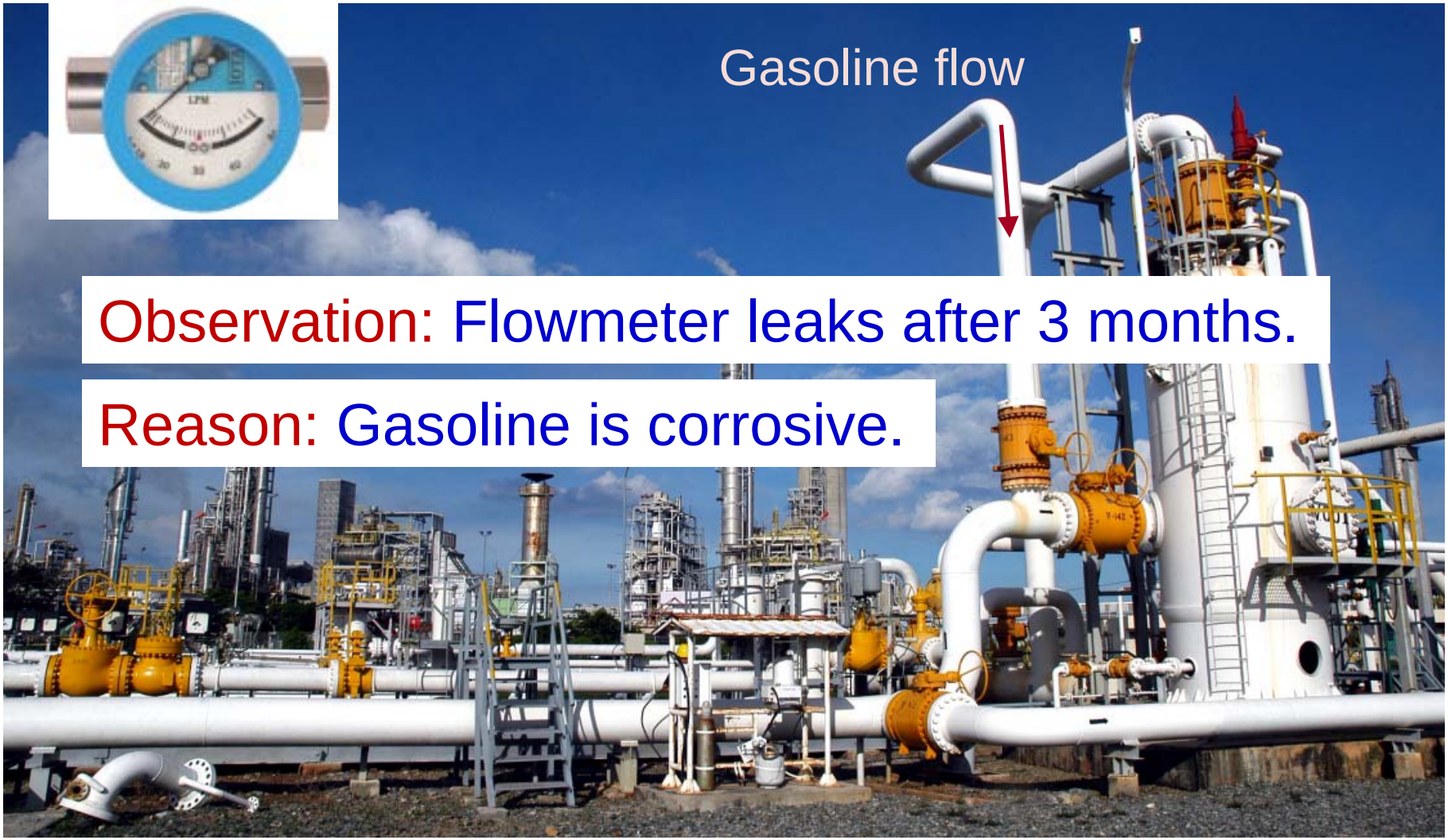
(1) The Hungry Bear Problem

Perceived problem: How to outrun the bear.

Right problem: How to outrun your companion!

How *hard* can "understanding the problem" be?

(2) Leaking Flowmeter



How *hard* can "understanding the problem" be?

(2) Leaking Flowmeter

Perceived problem:
How to prevent gasoline corrosion?

Solution: Build flowmeter with materials that would not be corroded by gasoline.

Reality: No such materials!

How *hard* can "understanding the problem" be?

(2) Leaking Flowmeter

Right problem:
How to prevent flowmeter leakage?

Solution: Replace flowmeter regularly.

How *hard* can "understanding the problem" be?

(3) Bargain Buy

~~$5.50~~

~~$5.00~~

$4.50

Observation: A particular brand is selling very well elsewhere but not in this store.

Perceived problem: Price not competitive.

Solution: Reduce the price.

Observation: Still not selling well.

Solution: Reduce the price further.

Observation: *Still* not selling well!

How *hard* can "understanding the problem" be?

(3) Bargain Buy

~~$5.50~~

~~$5.00~~

$4.50

Real problem: Item not prominently displayed!

Solution: Display item at prominent location.

Problem Definition

Guidelines for good problem definition

Clear, concise, unambiguous.

Capture all relevant information

- What are the inputs (data)?
- What are the outputs (unknowns)?
- What are the constraints?
- Connection/relationship between inputs and outputs?

Problem Definition

Verifying your problem definition

Is a solution possible?

Is a solution desirable?

Is there a trivial undesirable solution?

Do the constraints conflict?

Anything missing?

Anything extraneous?

Example #1: Tourist Problem

Verbal statement:

Given a group of tourists, each with a list of places to visit, schedule bus trips so that each tourist gets to visit all the places in his/her list.

Tourist	Places of Interest
Aaron	SZG, BG, JB
Betty	CG, JG, BG
Cathy	VC, SI, OR
David	JG, CG, OR
Evans	CG, JG, SZG
Frances	BG, SZG, JB
Gary	CG, OR
Harry	JG, CG

An instance of the tourist problem

Example #1: Tourist Problem

Problem definition (first attempt):

- Inputs: A list of tourists, each with a list of places to visit.
- Outputs: Bus trips.

Is this good enough?

Tourist	Places of Interest
Aaron	SZG, BG, JB
Betty	CG, JG, BG
Cathy	VC, SI, OR
David	JG, CG, OR
Evans	CG, JG, SZG
Frances	BG, SZG, JB
Gary	CG, OR
Harry	JG, CG

An instance of the tourist problem

Example #1: Tourist Problem

Revised problem definition:

- Inputs: A list of tourists, each with a list of places to visit.
- Outputs: A list of bus trips, each with a list of places to visit, and the tourists visiting.
- Constraints: Each tourist gets to visit all the places in his/her list.

Happy tourists

Example #1: Tourist Problem

Revised problem definition:

- Inputs: A list of tourists, each with a list of places to visit.
- Outputs: A list of bus trips, each with a list of places to visit, and the tourists visiting.
- Constraints: Each tourist gets to visit all the places in his/her list.

Is a solution possible/desirable?

A no-brainer solution:

One bus for each tourist!

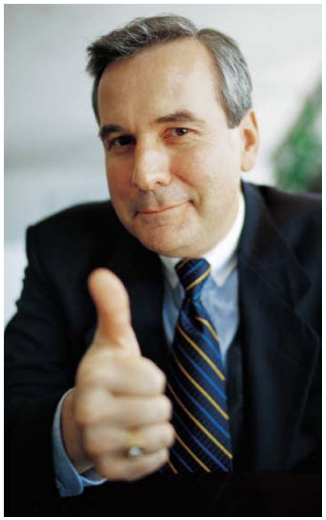
Unhappy boss

Example #1: Tourist Problem

Revised problem definition:

- Inputs: A list of tourists, each with a list of places to visit.
- Outputs: A list of bus trips, each with a list of places to visit, and the tourists visiting.
- ~~Constraints: Each tourist gets to visit all the places in his/her list.~~
- Constraints:
  - Each tourist gets to visit all the places in his/her list.
  - Each tourist visits at most one place per day.
  - Fewest possible days.
  - At most one bus trip to each place.

Happy tourists



Happy boss

Example #1: Tourist Problem

- Revised constraints:
  - Each tourist gets to visit all the places in his/her list.
  - Each tourist visits at most one place per day.
  - Fewest possible days.
  - At most one bus trip to each place.
  - Each bus takes at most 4 tourists.

Do the constraints conflict?

Tourist	Places of Interest
Aaron	SZG, BG, JB
Betty	CG, JG, BG
Cathy	VC, SI, OR
David	JG, CG, OR
Evans	CG, JG, SZG
Frances	BG, SZG, JB
Gary	CG, OR
Harry	JG, CG

Lessons Learned

Verbal statements can be vague.

Constraints may be implicit.

Without constraints, may get trivial but undesirable solution.

When constraints conflict, no solution.

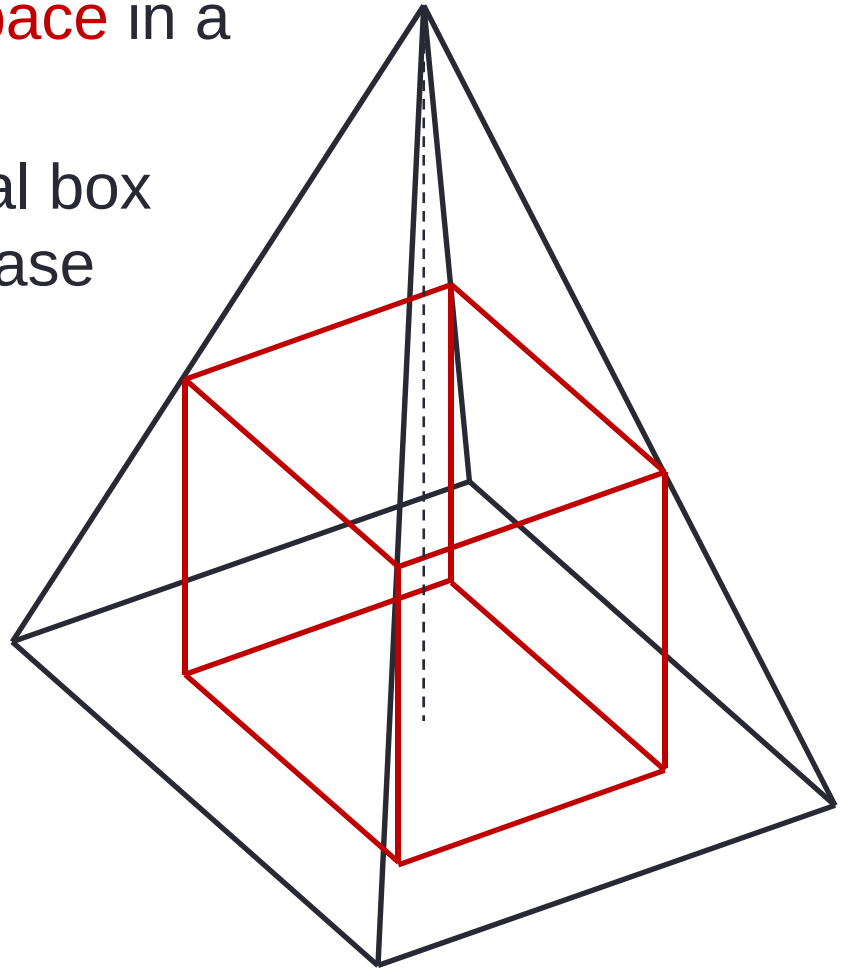
Use Polya's questions to guide your reasoning.

Example #2: Cube in Pyramid

Verbal statement:

What is the largest cubical space in a square-base pyramid?

(Or, what is the largest cubical box we can put inside a square-base pyramid?)



Example #2: Cube in Pyramid

Problem definition:

- Inputs (data):
  - Square-base pyramid with base width *b* and height *h*.
- Outputs (unknowns):
  - Cube of width *w*.
- Constraints (conditions):
  - Base of cube rests on base of pyramid.
  - Top 4 corners of cube touch some surfaces of the pyramid.

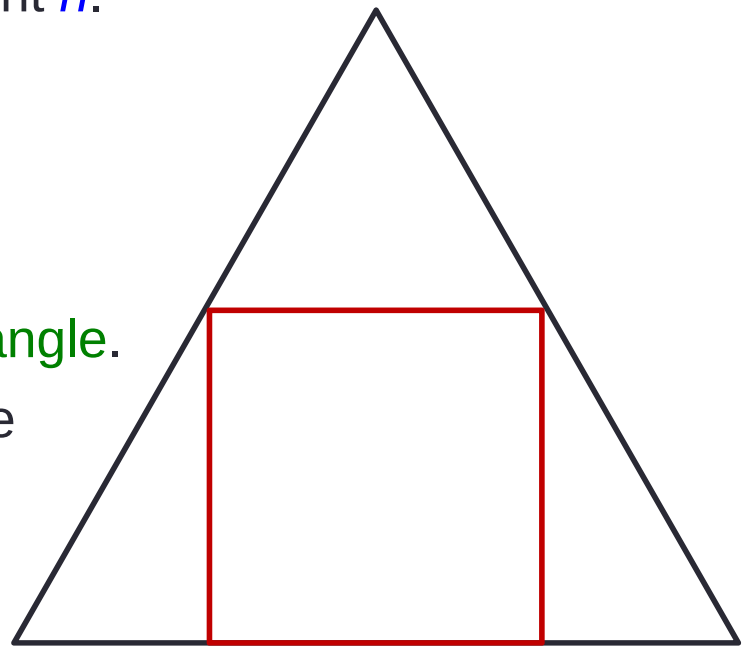
Too hard!

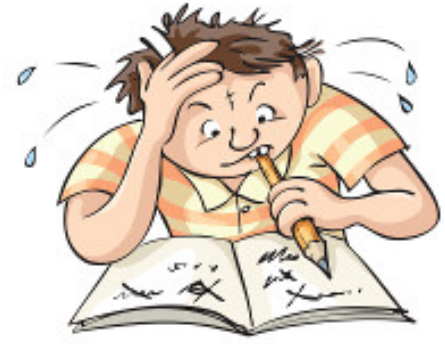
What do you do when a problem looks too hard?

Example #2: Cube in Pyramid

Simplified problem definition in 2D:

- Inputs (data):
  - Triangle with base width *b* and height *h*.
- Outputs (unknowns):
  - Square of width *w*.
- Constraints (conditions):
  - Base of square rests on base of triangle.
  - Top 2 corners of square touch some edges of the triangle.



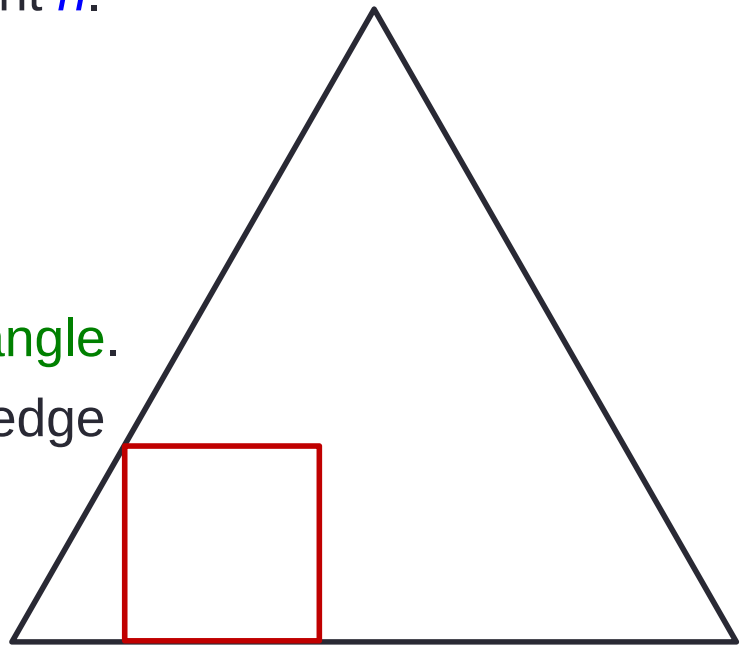


Still too hard!

Example #2: Cube in Pyramid

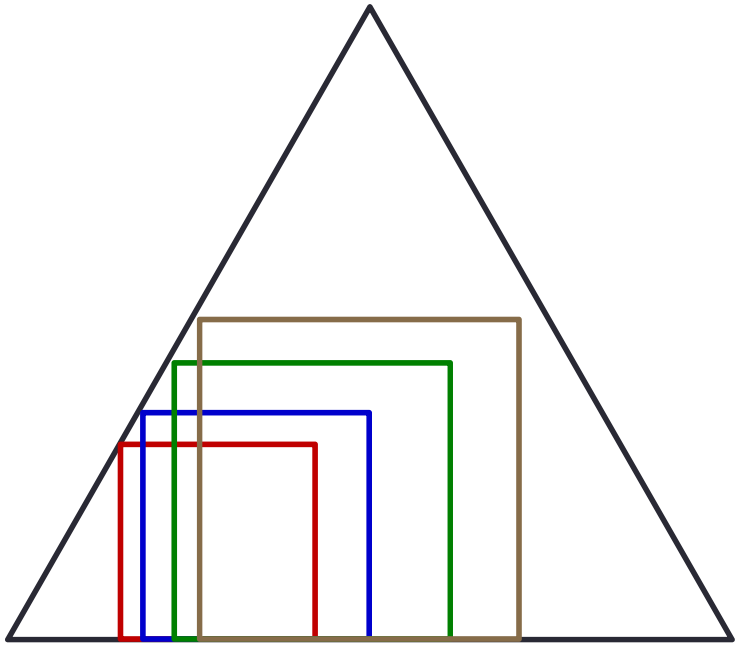
More simplified problem definition in 2D:

- Inputs (data):
  - Triangle with base width *b* and height *h*.
- Outputs (unknowns):
  - Square of width *w*.
- Constraints (conditions):
  - Base of square rests on base of triangle.
  - Top 1 corner of square touches an edge of the triangle.



Example #2: Cube in Pyramid

The more simplified problem is so simple
it has many simple solutions!



Can you see the solution of the original problem now?

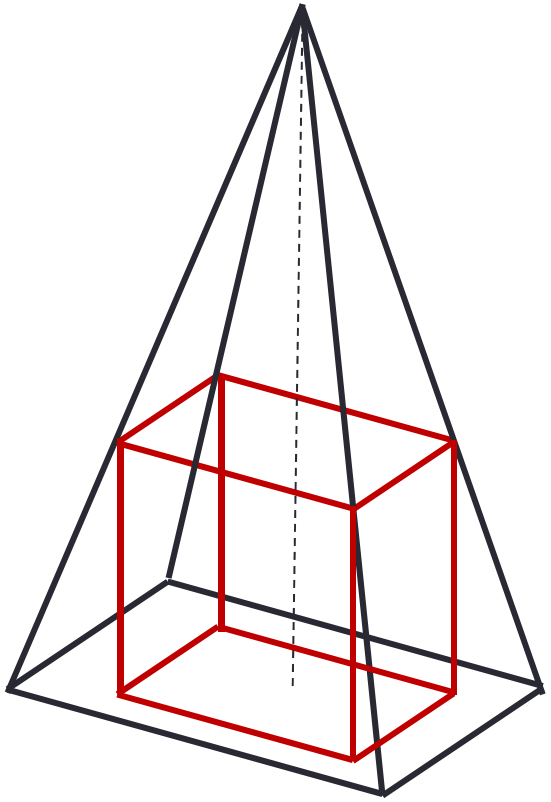
Example #2: Cube in Pyramid

Polya heuristic:

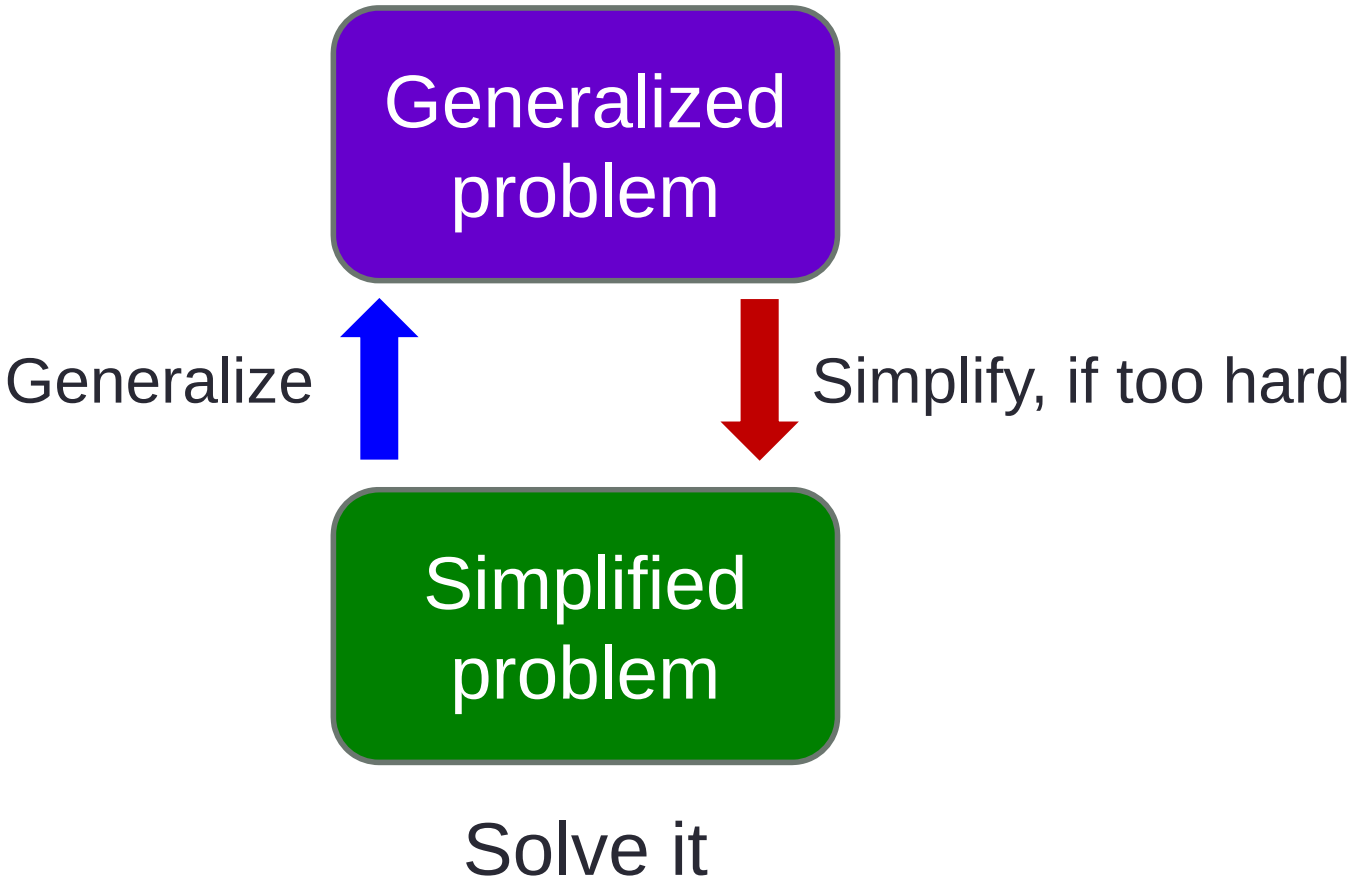
After solving simpler problem, generalize.

Generalized problem:

What is the largest rectangular block in a rectangular-base pyramid?



Lesson Learned




SUMMARY

Computational Thinking

Decomposition

Abstraction

The process Computer Scientists and Programmers use when solving problems like these is **Computational Thinking**. Whatever the problem, their job is to determine instructions and actions that the computer can understand and put to use to solve it. In order to work in the language that computers can translate, Programmers first **break the problem down into components, or smaller pieces**. They **look for patterns within and/or between these subproblems** in order to maximize efficiency by programming shortcuts that can be repeated and combined. They also **think in terms of generalized problem variables so that the main components become clear, and distracting, unimportant information can be minimized**. This allows for one variable to represent many and can allow for the modeling and simulation of a variety of situations. Finally, their job is to **create clear, unambiguous, ordered and finite instructions, or recipes, that tell the computer what to do and in what order it needs to be done**.

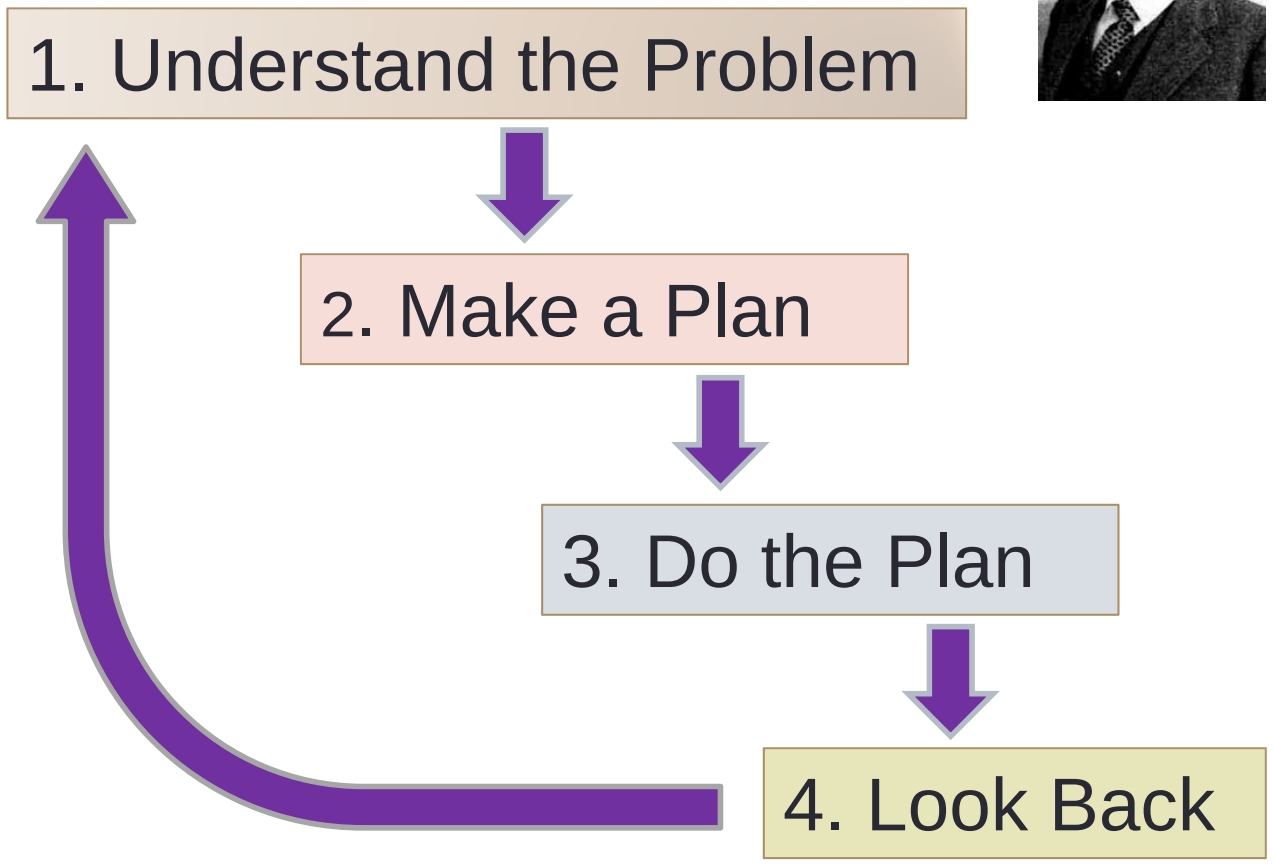
Algorithm Design

Pattern Recognition

Computational Thinking

ATTITUDES	SKILLS
Making mistakes	Decomposition
Perseverance	Pattern Recognition
Imagination	Abstraction
Collaboration	Algorithm Design

Steps to Problem Solving



Formulate problems for computer solutions.

Logically organize and analyse data.

Represent data through abstraction.

Develop algorithmic solution.

Identify most efficient solution.

Generalize problem solving process to other areas.

Communicate above processes.

End of File