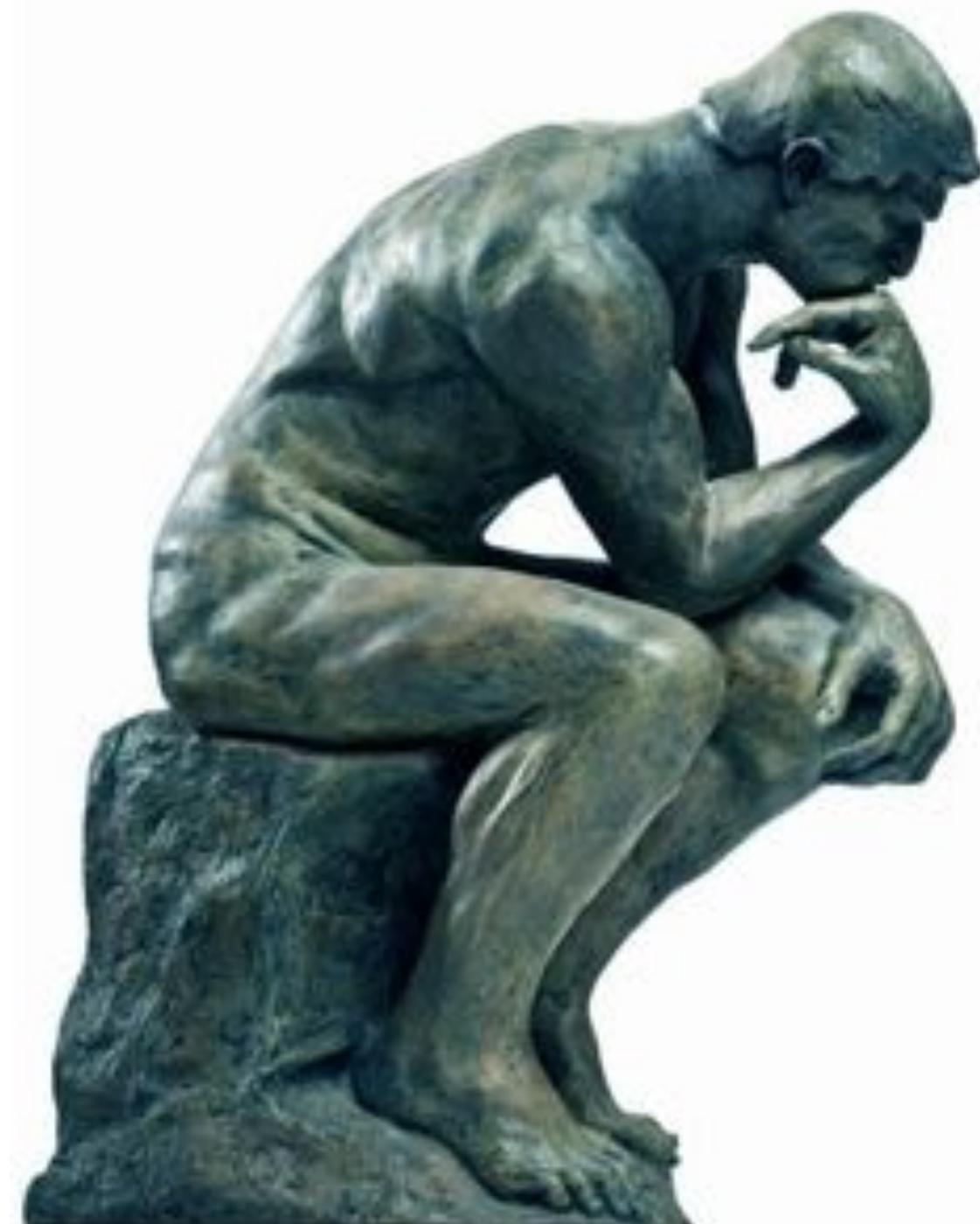
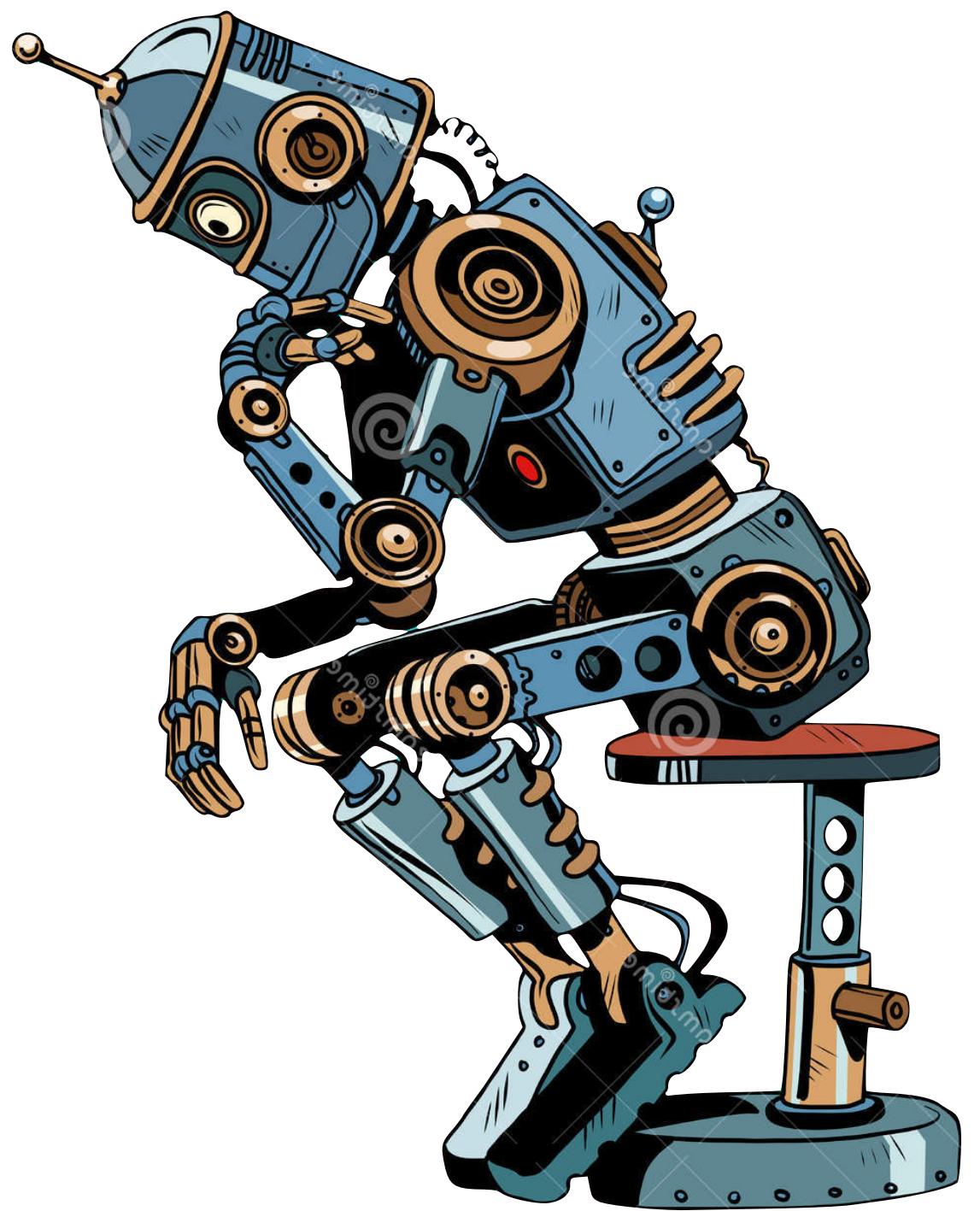


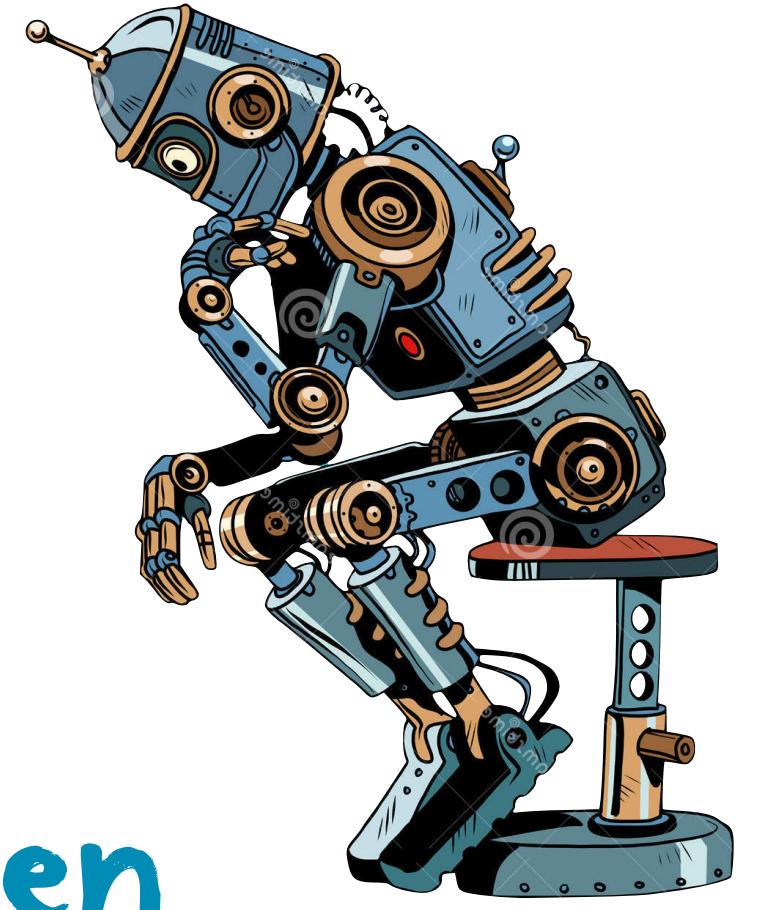
algorithms and computational thinking



welcome!

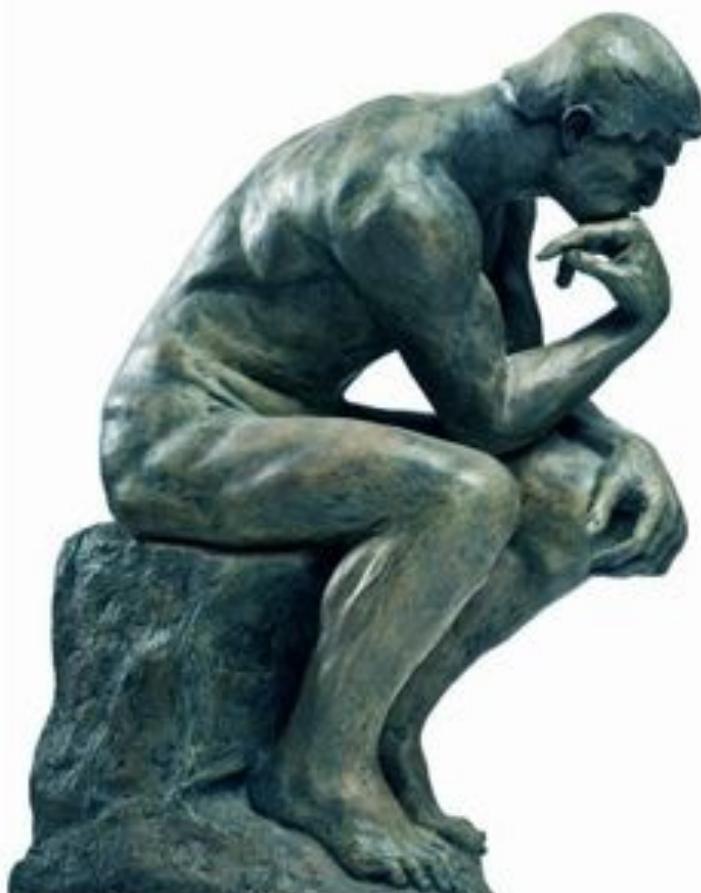


algorithms and computational thinking



enseignement comodal : une partie des étudiant·e·s en salle de cours, une autre partie en ligne suivant le cours diffusé en direct ou en différé

dual mode teaching : some of the students in the classroom, some online following the live or recorded course



algorithms and
computational
thinking



dual mode teaching each week

1/3 of the students follow the course
in the class room

2/3 of the students follow the course
online via streaming

lectures: Amphimax 351 ←

exercises: Amphipôle 140 + 146 ←

algorithms and computational thinking



dual mode teaching

Septembre

Lu	Ma	Me	Je	Ve	Sa	Di
	01	02	03	04	05	06
07	08	09	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

Octobre

Lu	Ma	Me	Je	Ve	Sa	Di
			01	02	03	04
05	06	07	08	09	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

Novembre

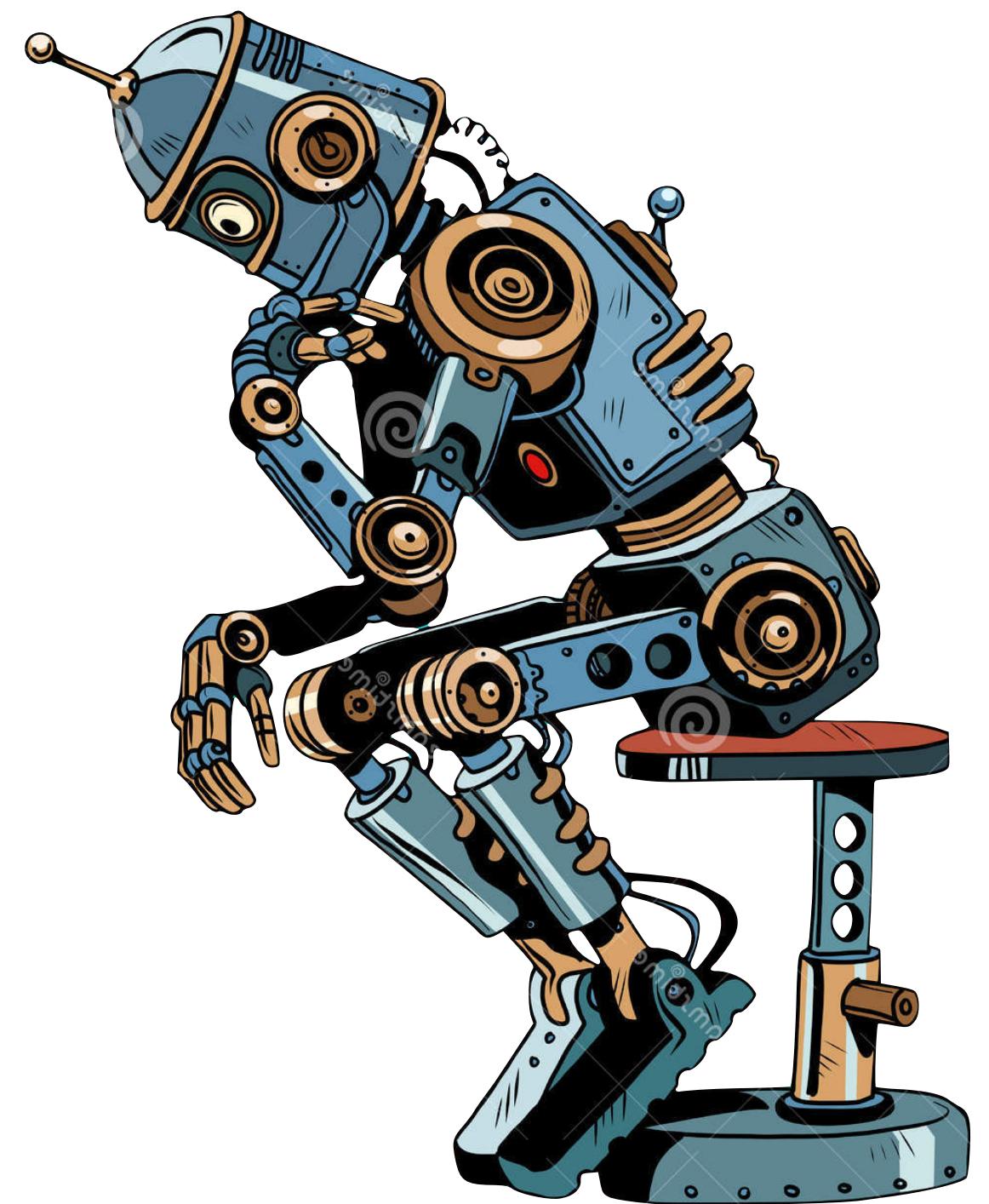
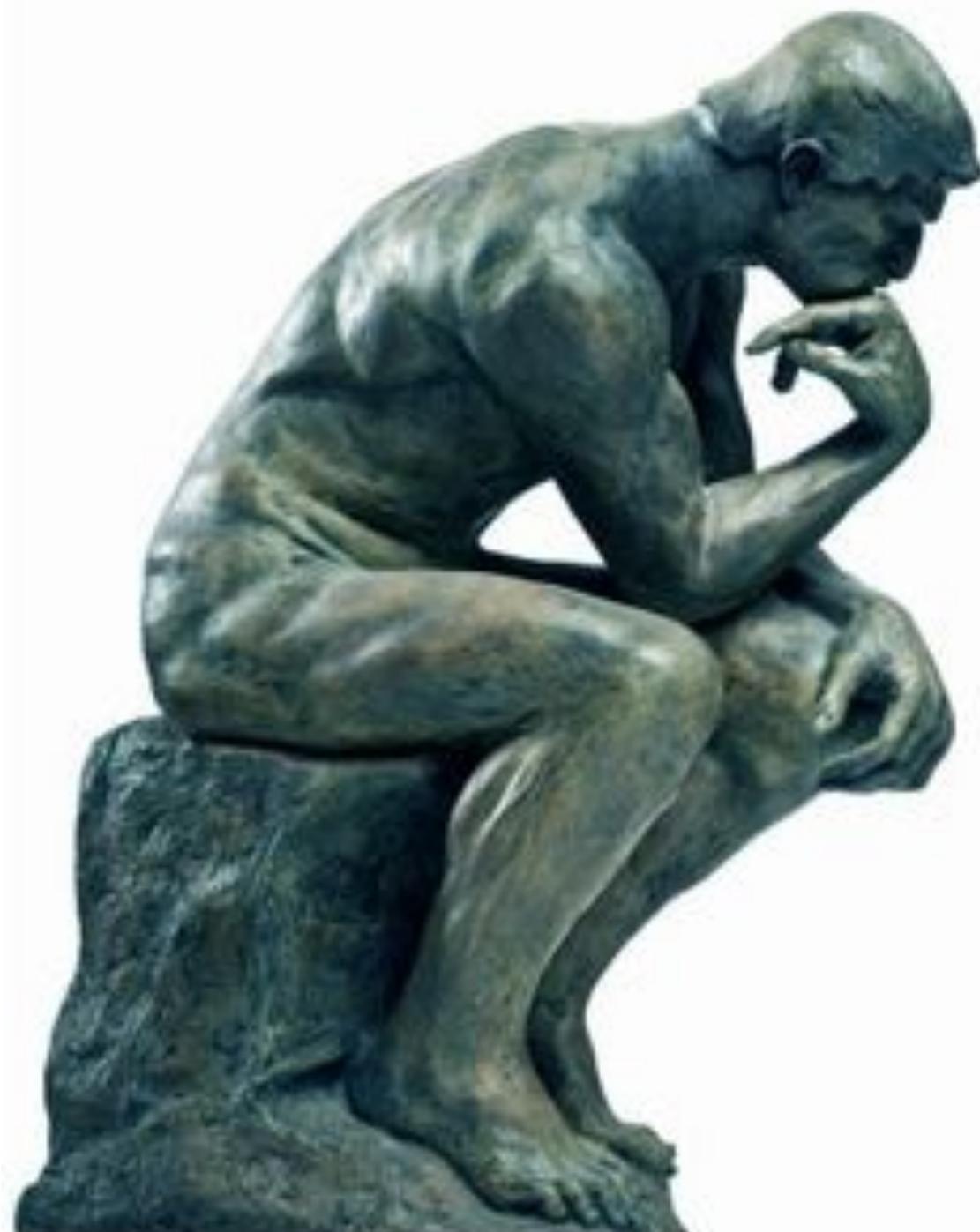
Lu	Ma	Me	Je	Ve	Sa	Di
				01		
02	03	04	05	06	07	08
09	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

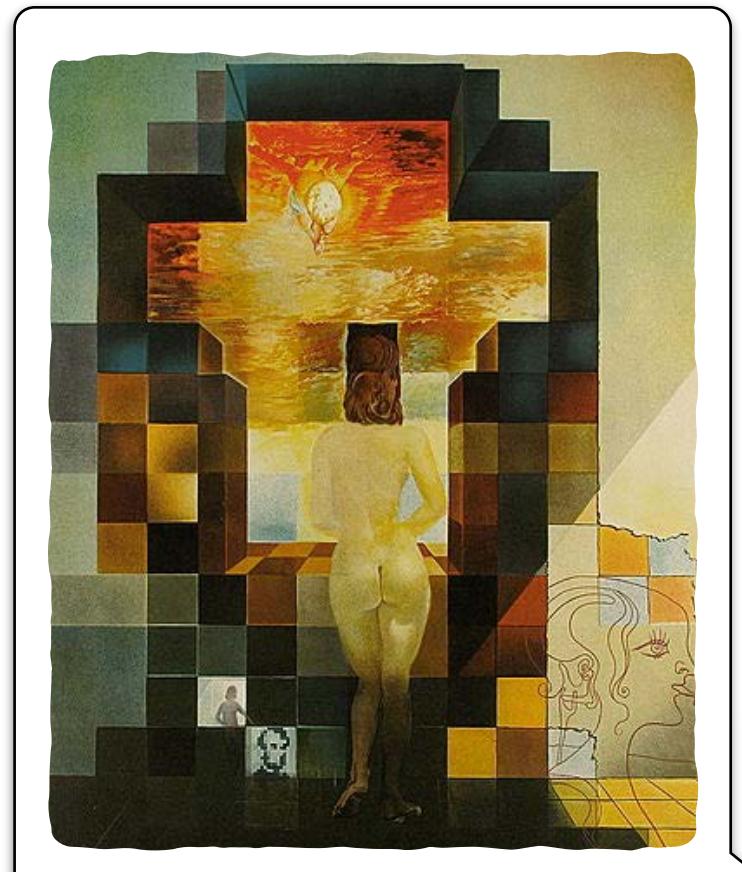
Décembre

Lu	Ma	Me	Je	Ve	Sa	Di
	01	02	03	04	05	06
07	08	09	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

algorithms and computational thinking

course
overview



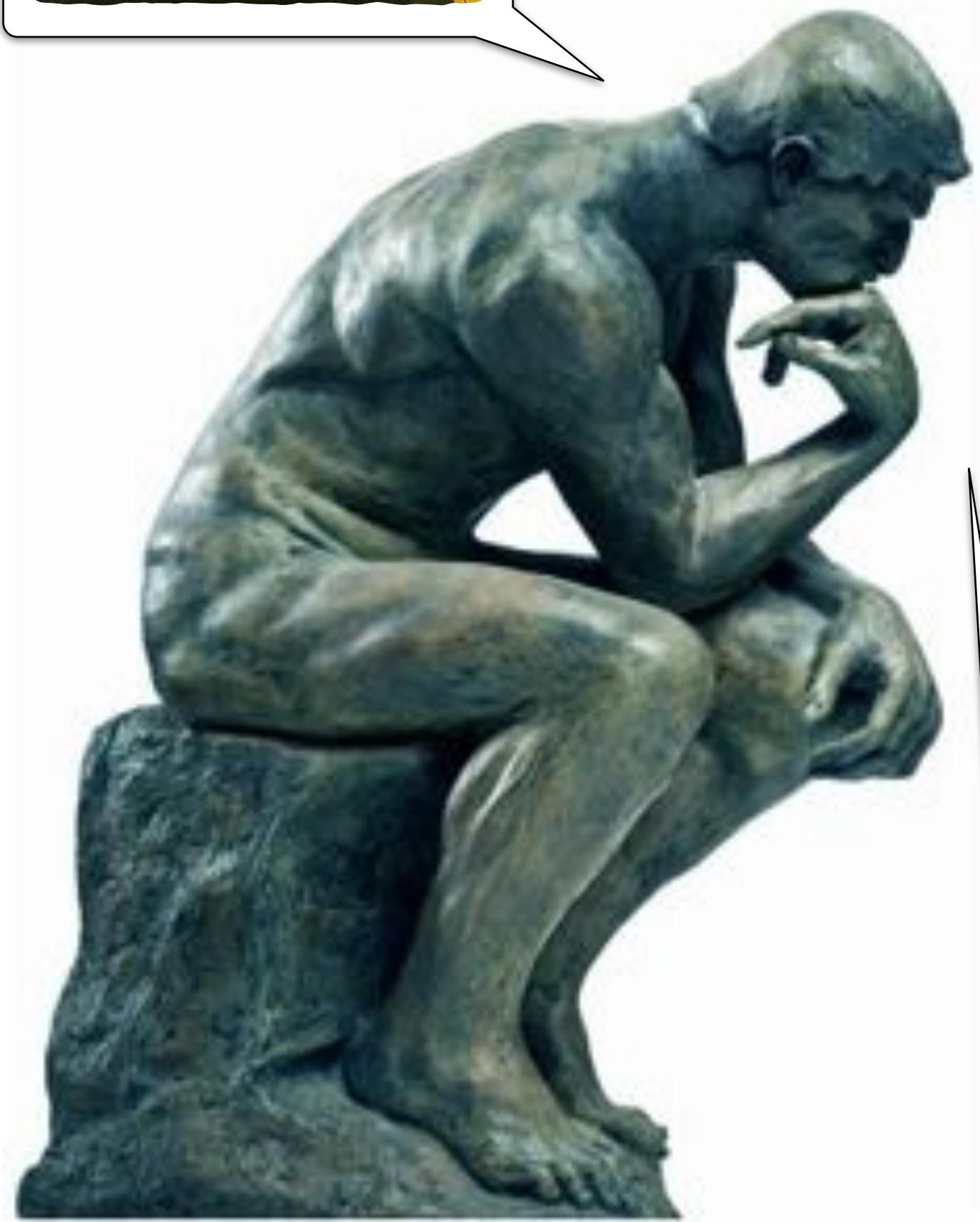


let's talk

$$x^n + y^n = z^n$$

1111001101010011

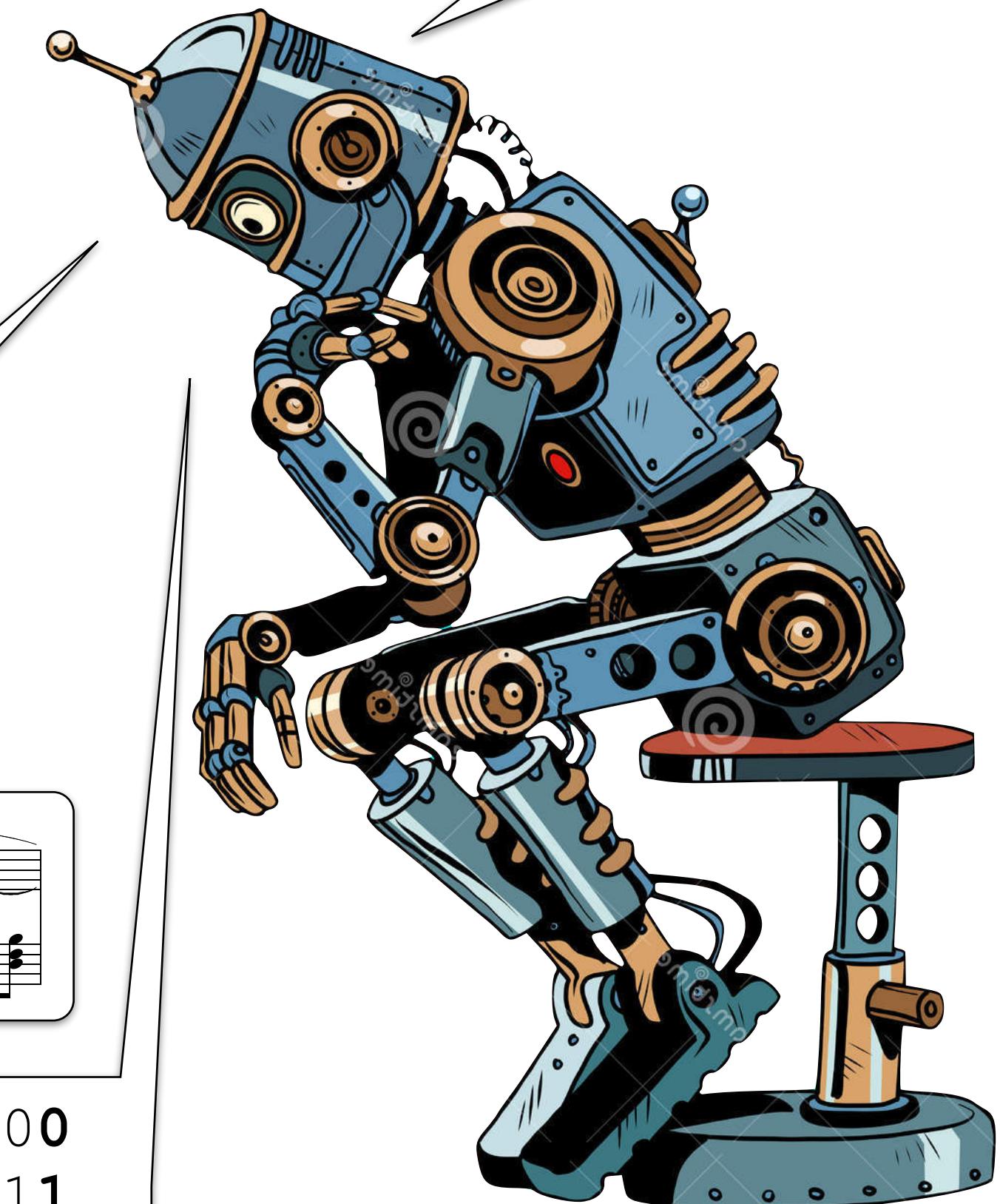
00100101
00101011
00010010
10100100
11001101
00111001
11110011
01010011

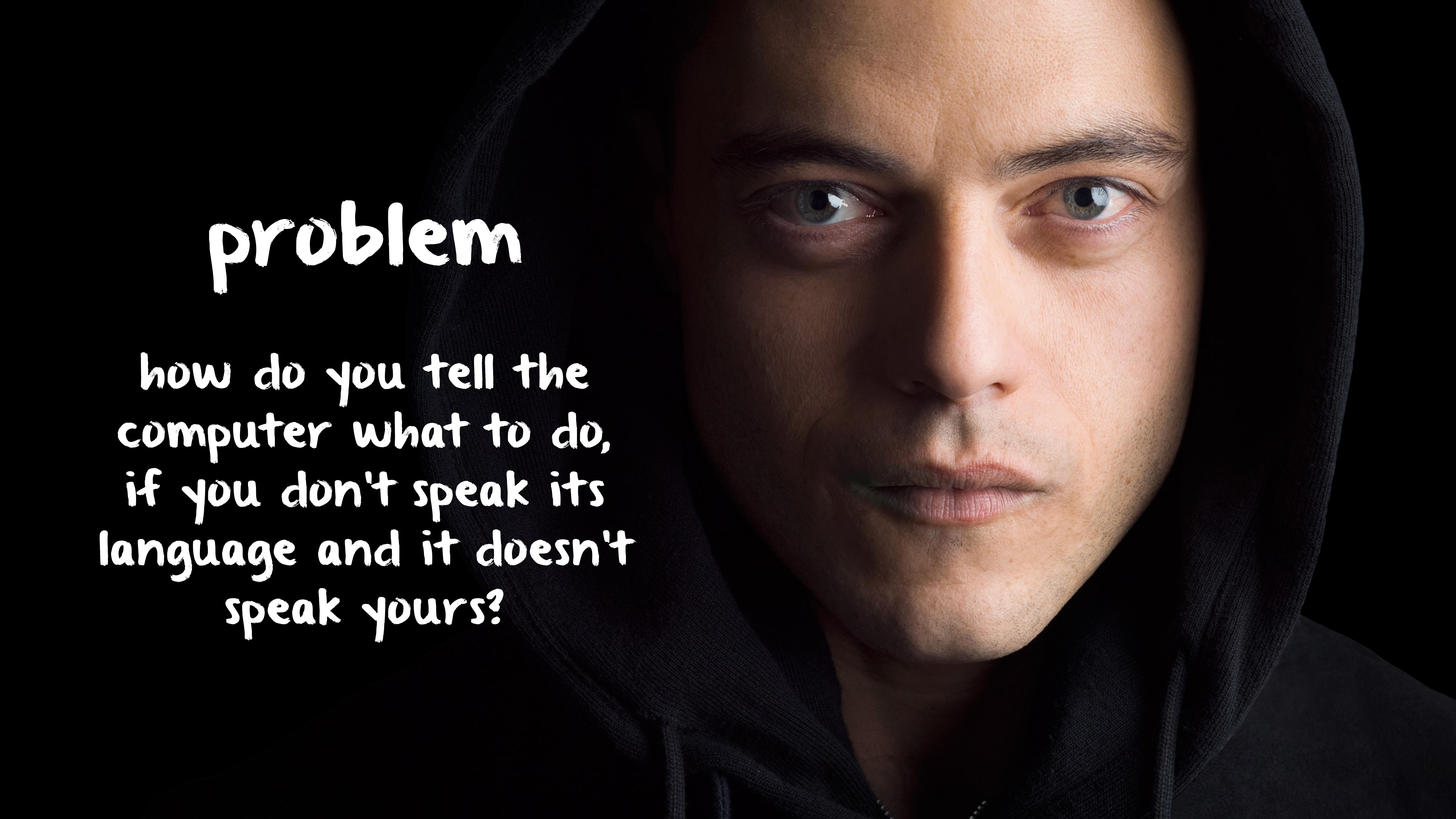


0010010100101011
1100110100111001
1111001101010011



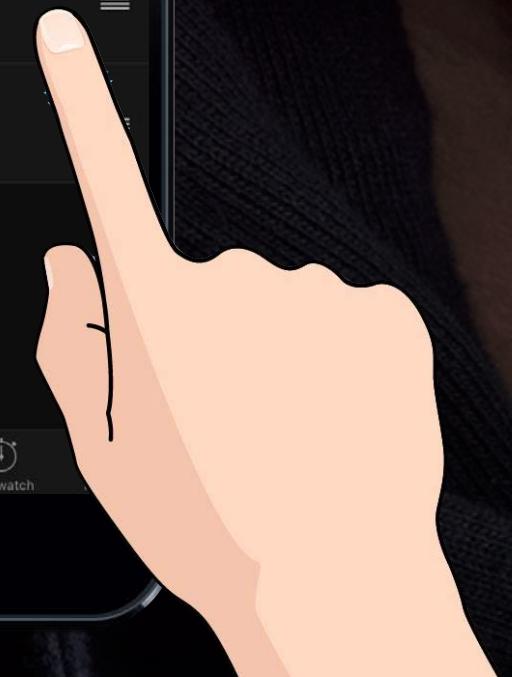
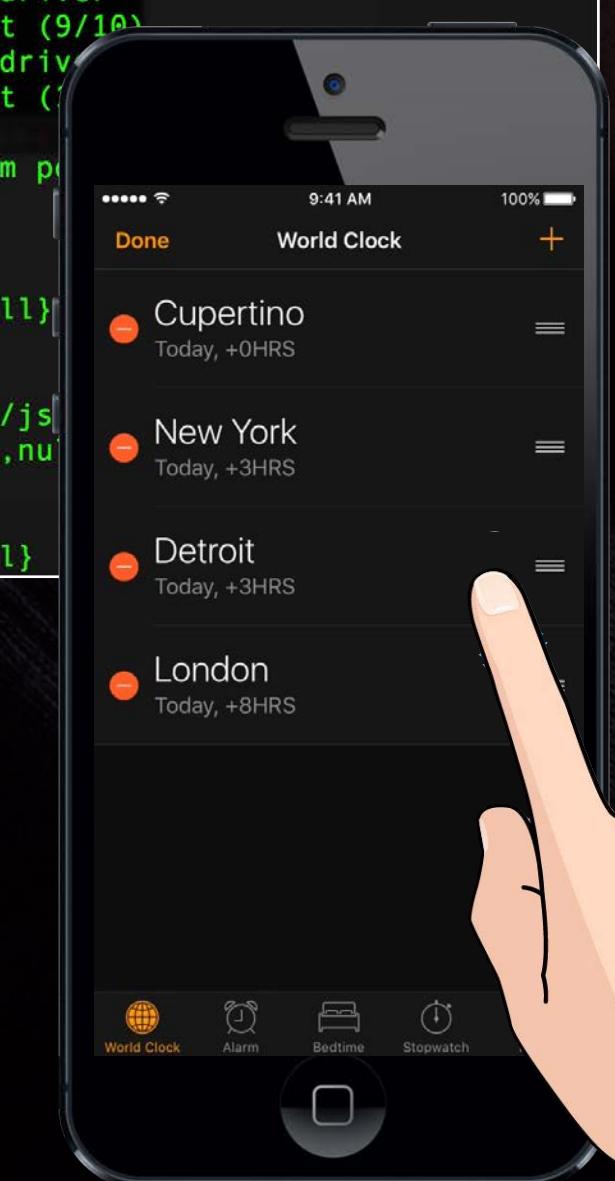
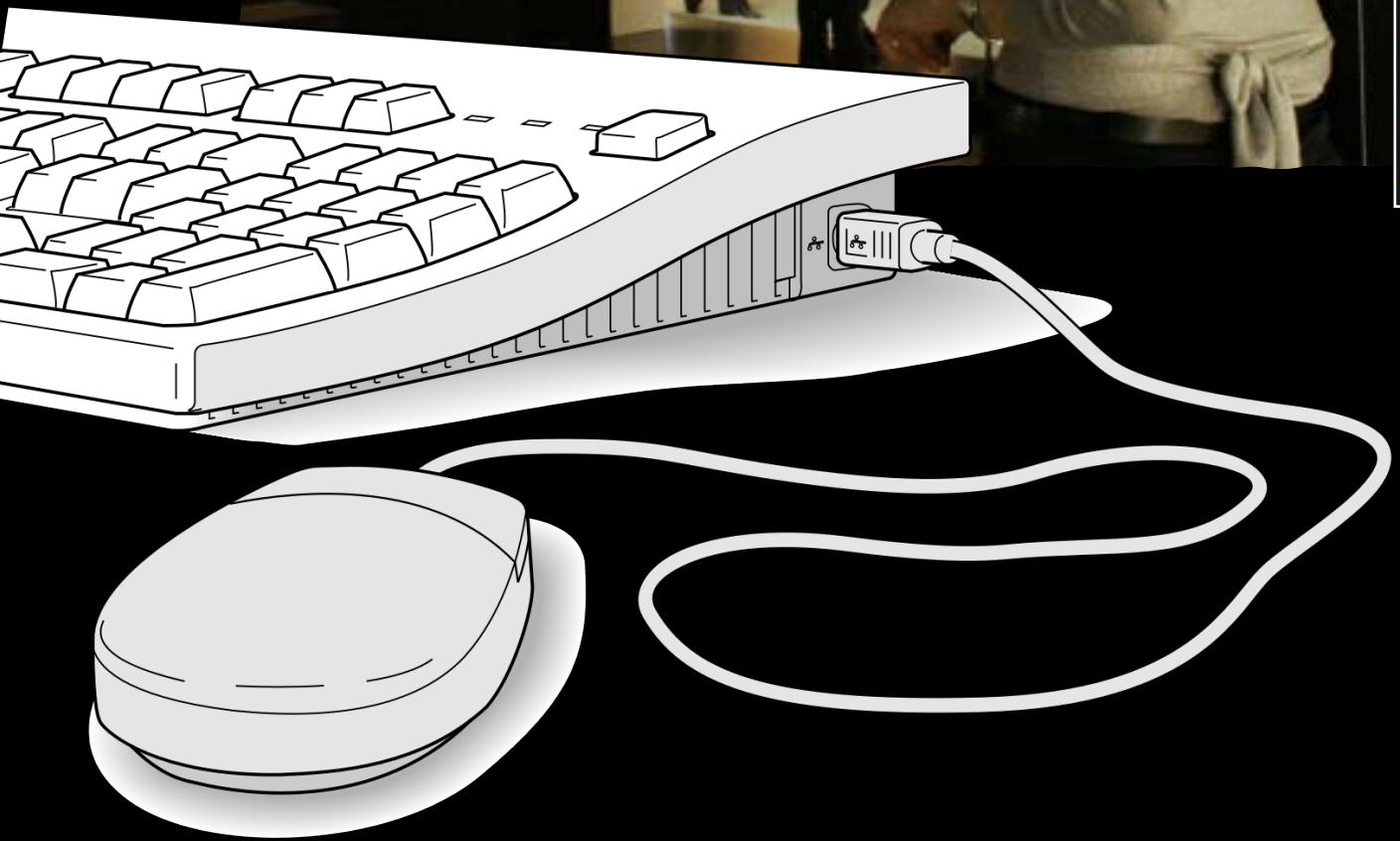
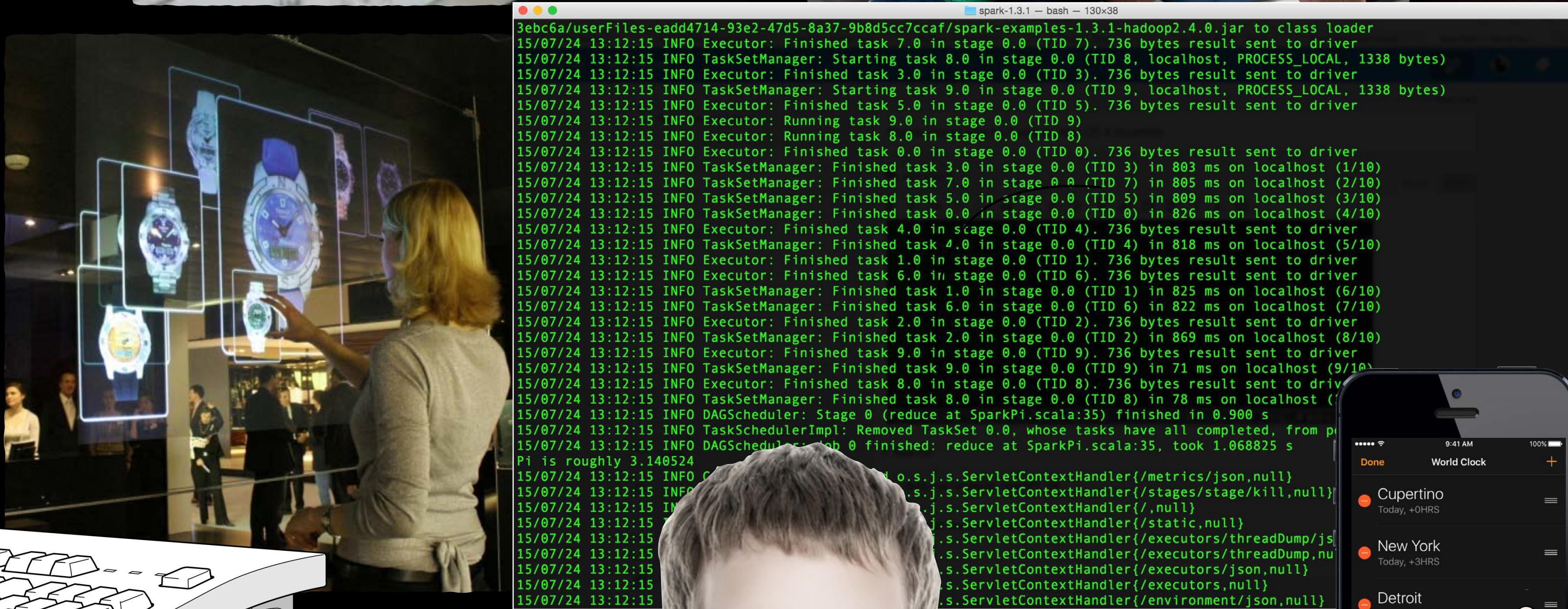
0010010100101011001001010100100
11001101001110011111001101010011



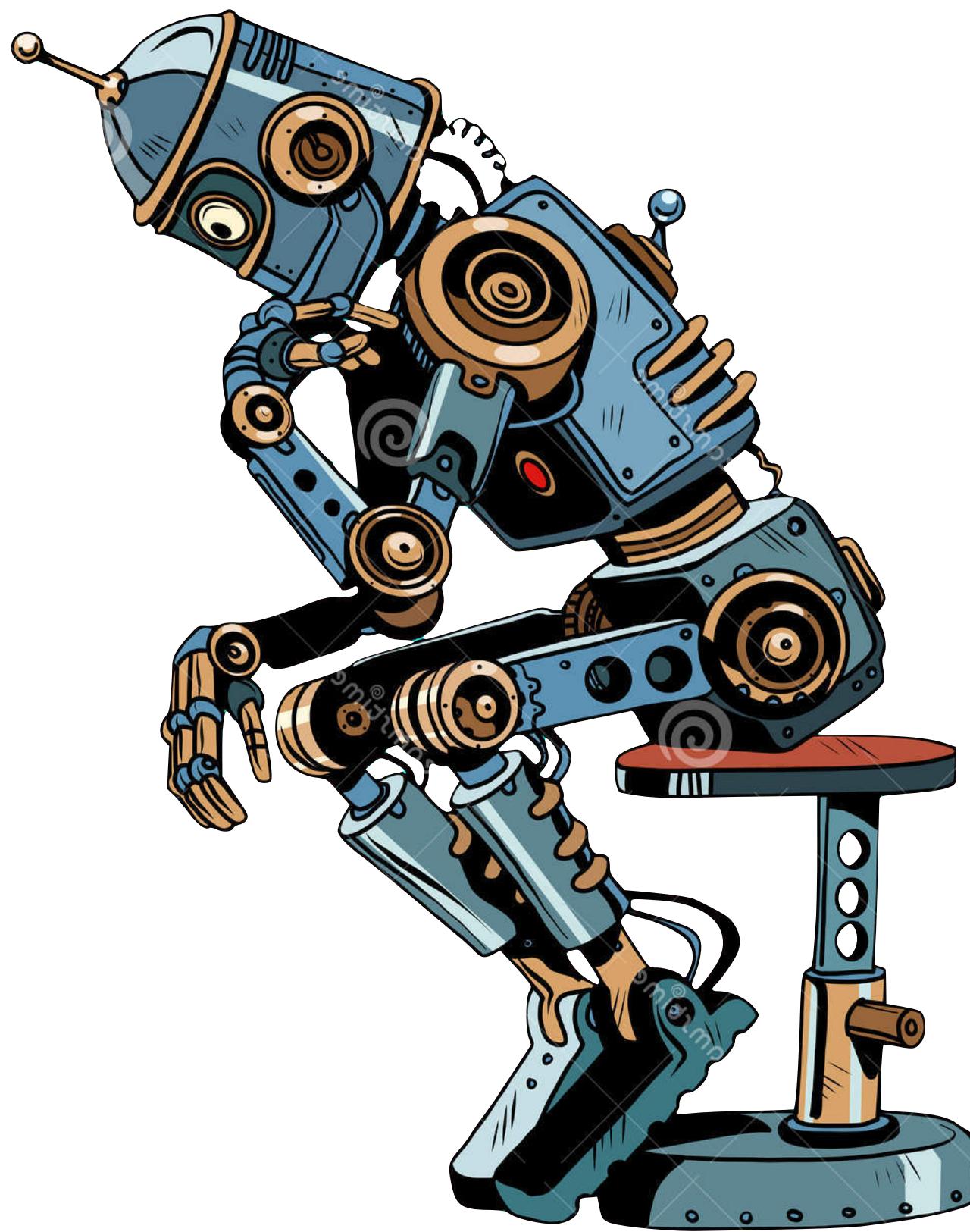
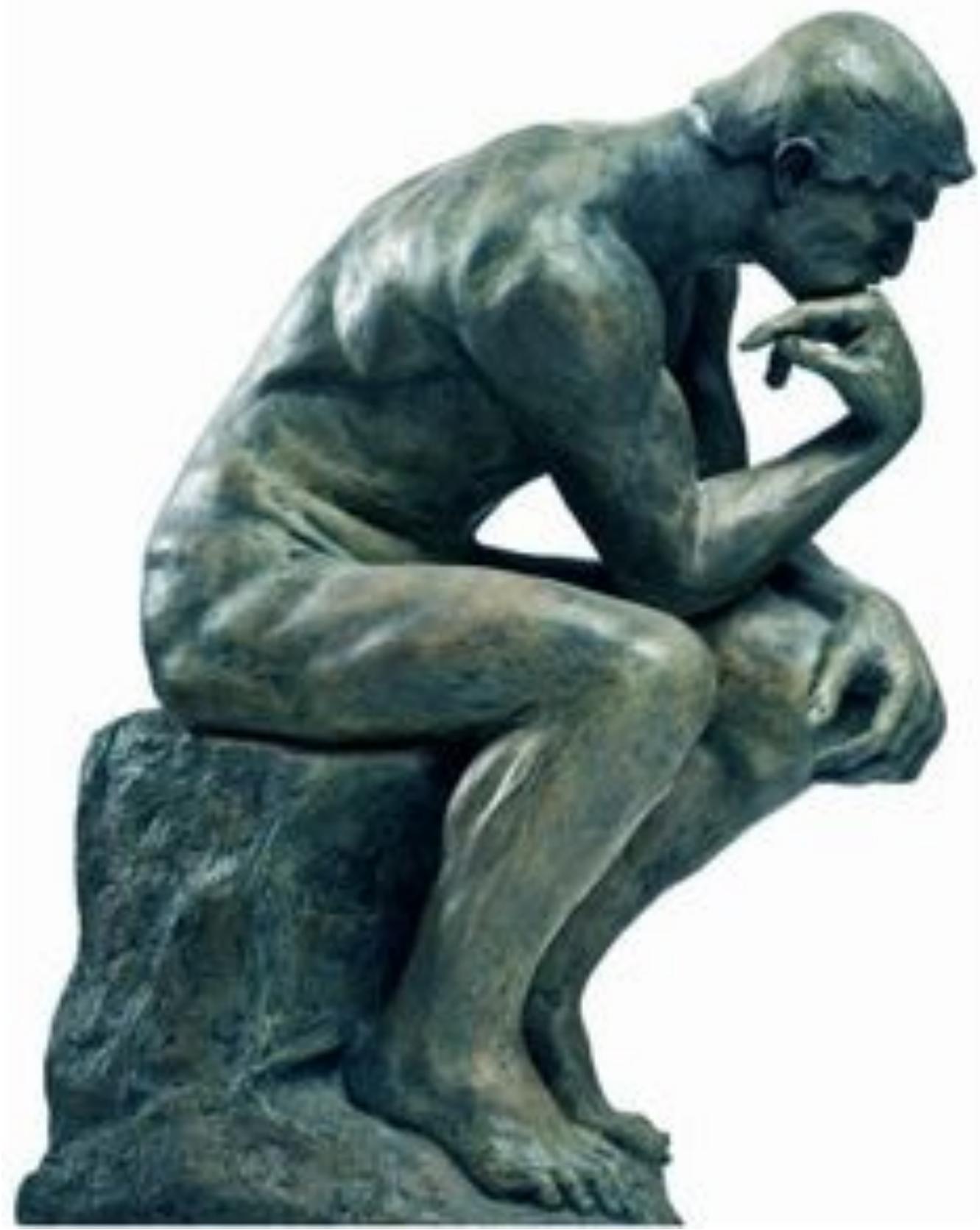


problem

how do you tell the
computer what to do,
if you don't speak its
language and it doesn't
speak yours?



approach of this course



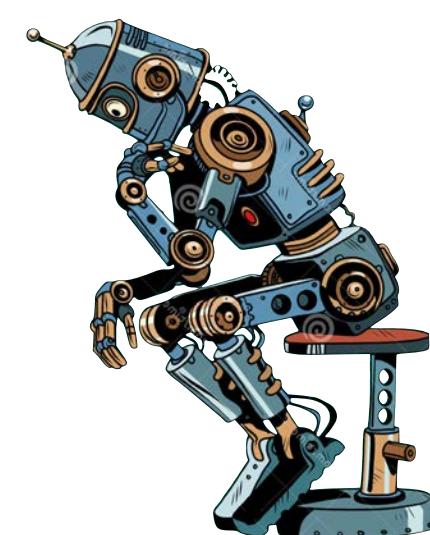
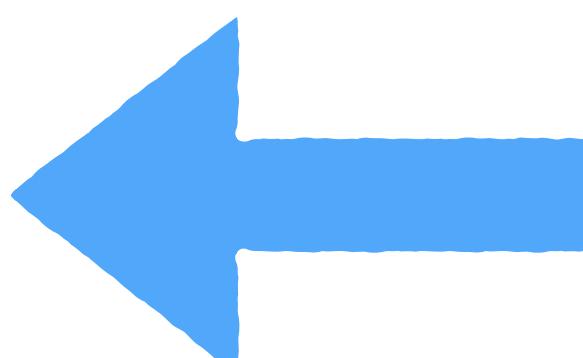
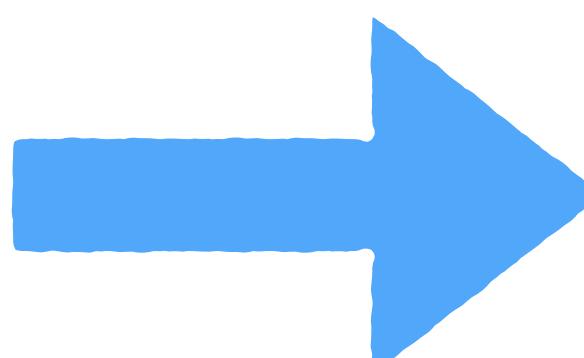
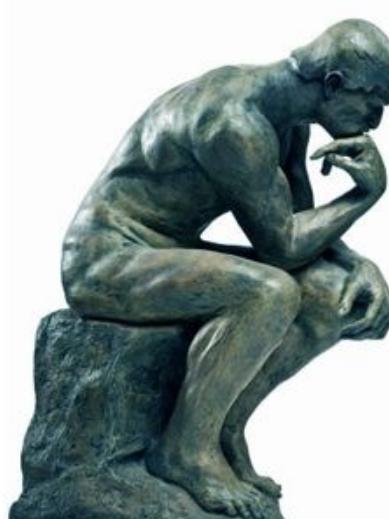
approach of this course

think precisely in
computational terms

express problems formally and
solutions algorithmically

use high-level
programming languages

use tools to develop,
and debug programs



computer science

theoretical and practical study
of how to design and use
computer-based systems

computer science aims at devising automated algorithmic processes
and computer-based systems that can run in a scalable manner

computational thinking

thought processes involved in formulating problems and expressing their solutions so that a computer can execute them

such solutions are expressed in terms of algorithms, which are in turn written in some programming language compiled and executed on some computer-based system

computer science

(design and use computer-based systems)

computational thinking

(only use computer systems)

computational thinking \subset *computer science*

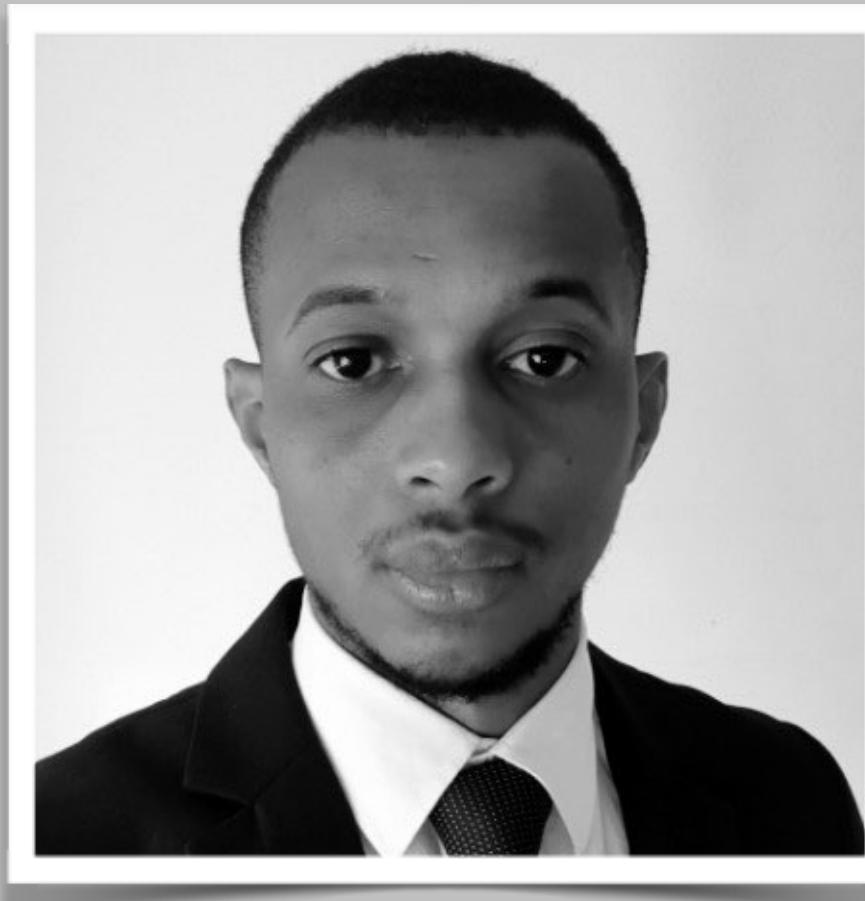
computer science $\not\subset$ *computational thinking*

teaching staff



Benoît
Garbinato

professor



Alpha
Diallo

assistant

student assistants



Gaëtan



Johannes



Maeva



Nathan



Richard



Yasser

course

learn a set of
thinking skills and
practical methods
to formulate and
solve problems using
algorithms and
computing devices

objective



course

objective

what's a computer?

what's an algorithm?

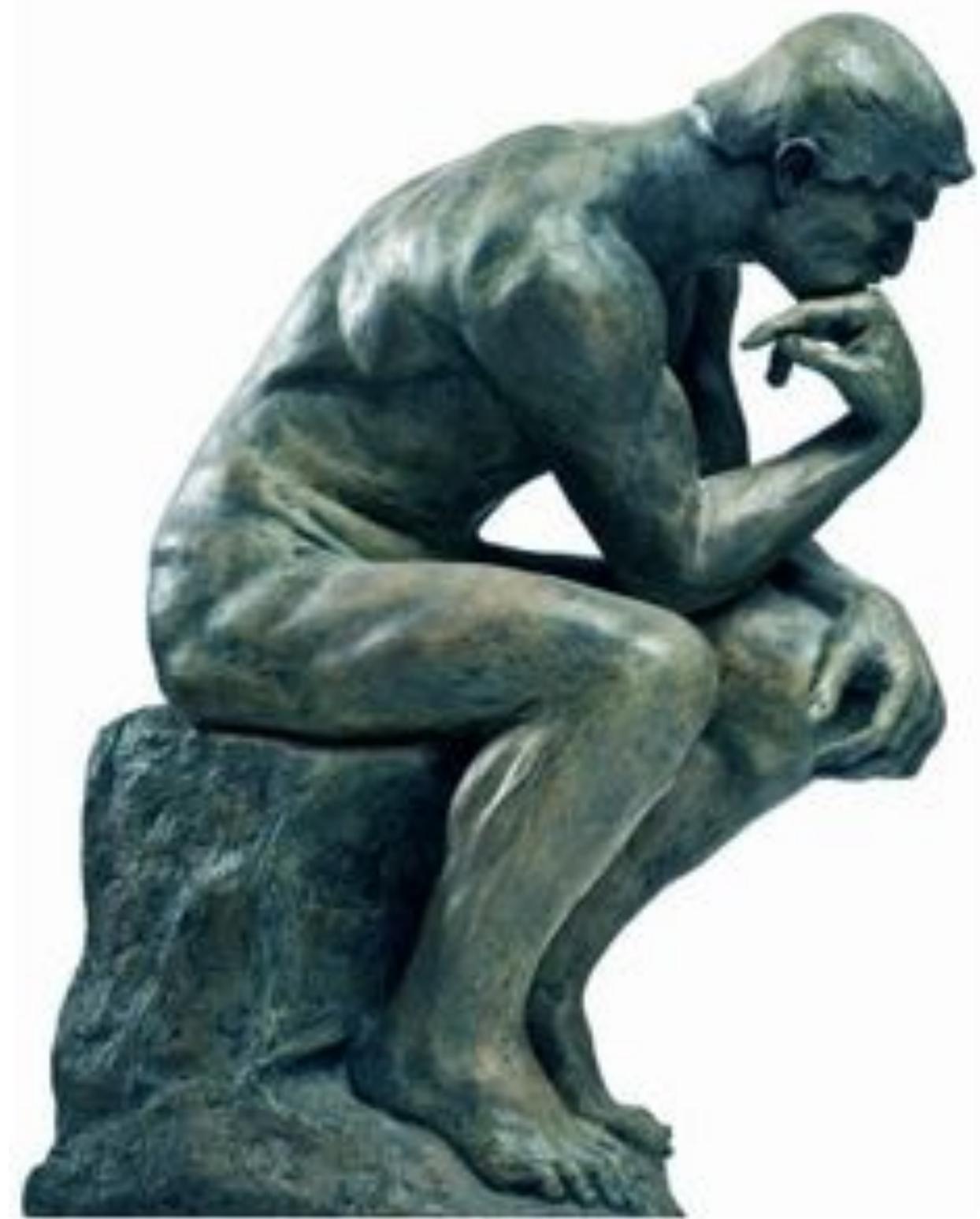
what's a compiler?

what's programming?

etc...

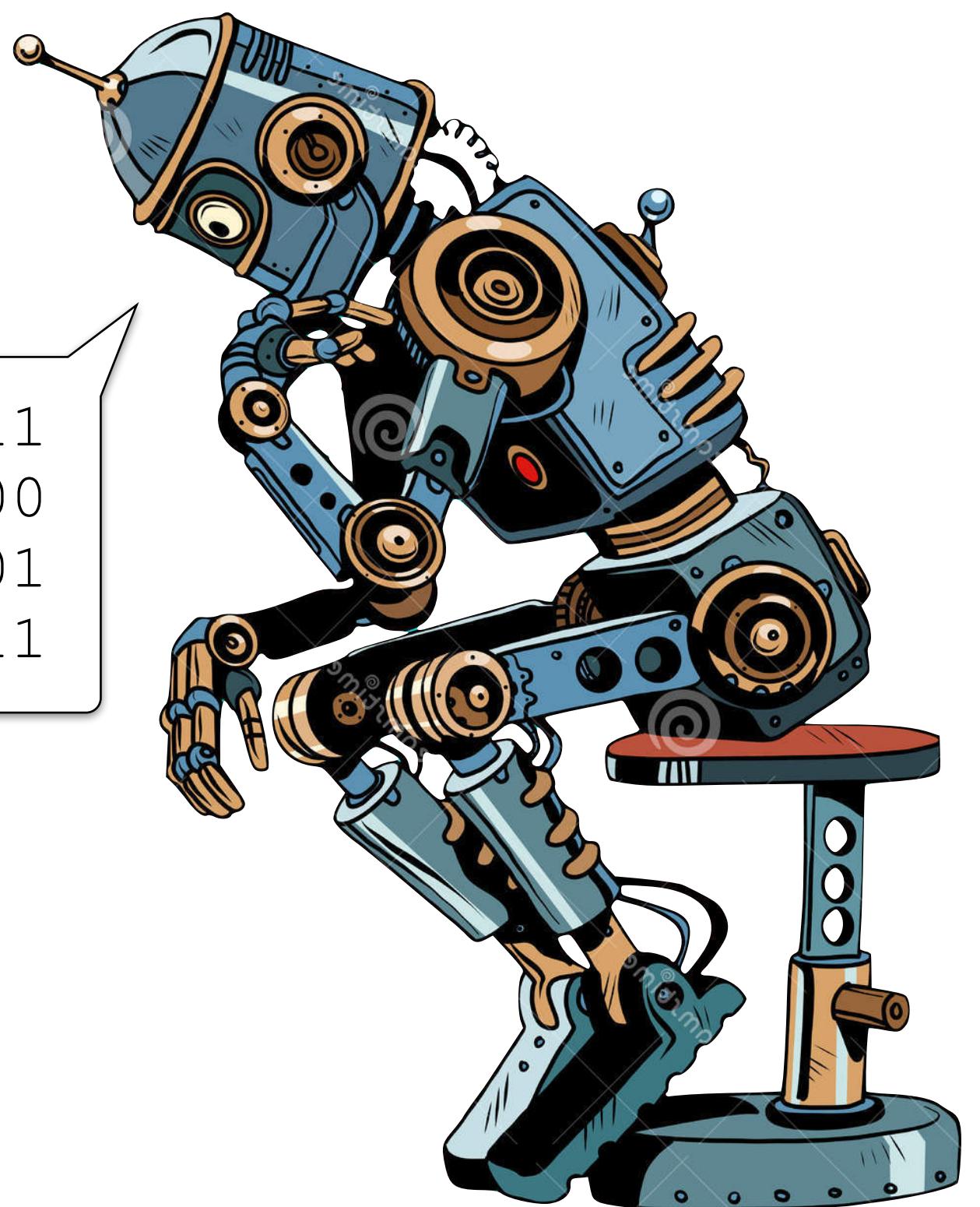


content & approach



$i \leftarrow i + 1$

```
0010010100101011  
0001001010100100  
1100110100111001  
1111001101010011
```



content & approach

algorithms

$i \leftarrow i + 1$

software

$i = 0$
 $i = i + 1$

hardware

0010010100101011
0001001010100100
1100110100111001
1111001101010011

content & approach

algorithms

$i \leftarrow i + 1$

your software

$i = 0$
 $i = i + 1$

{runtime | interpreter} + Libraries

operating system

} system
software

hardware

0010010100101011
0001001010100100
1100110100111001
1111001101010011

what are the
benefits of this
course?

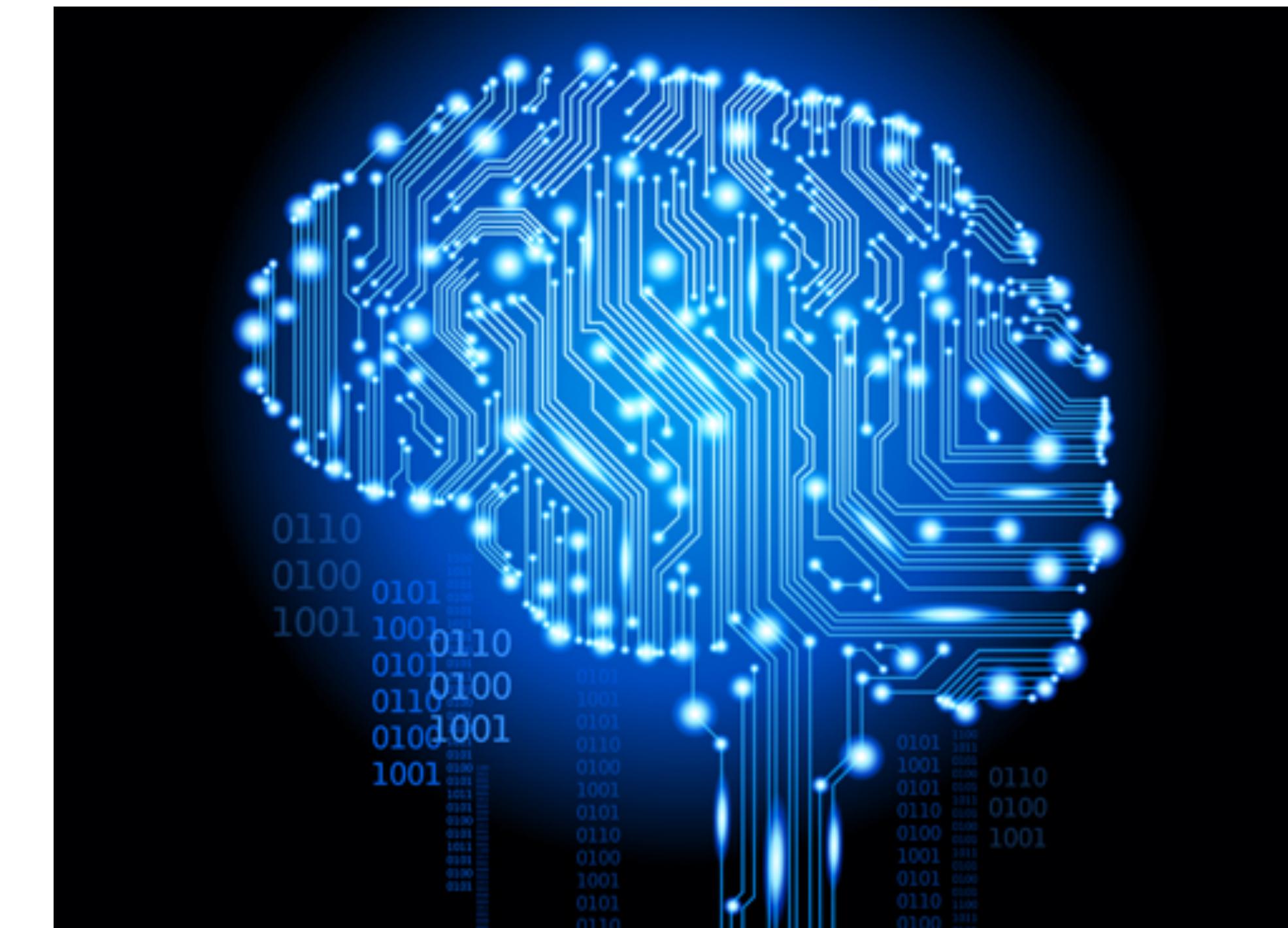
direct benefits

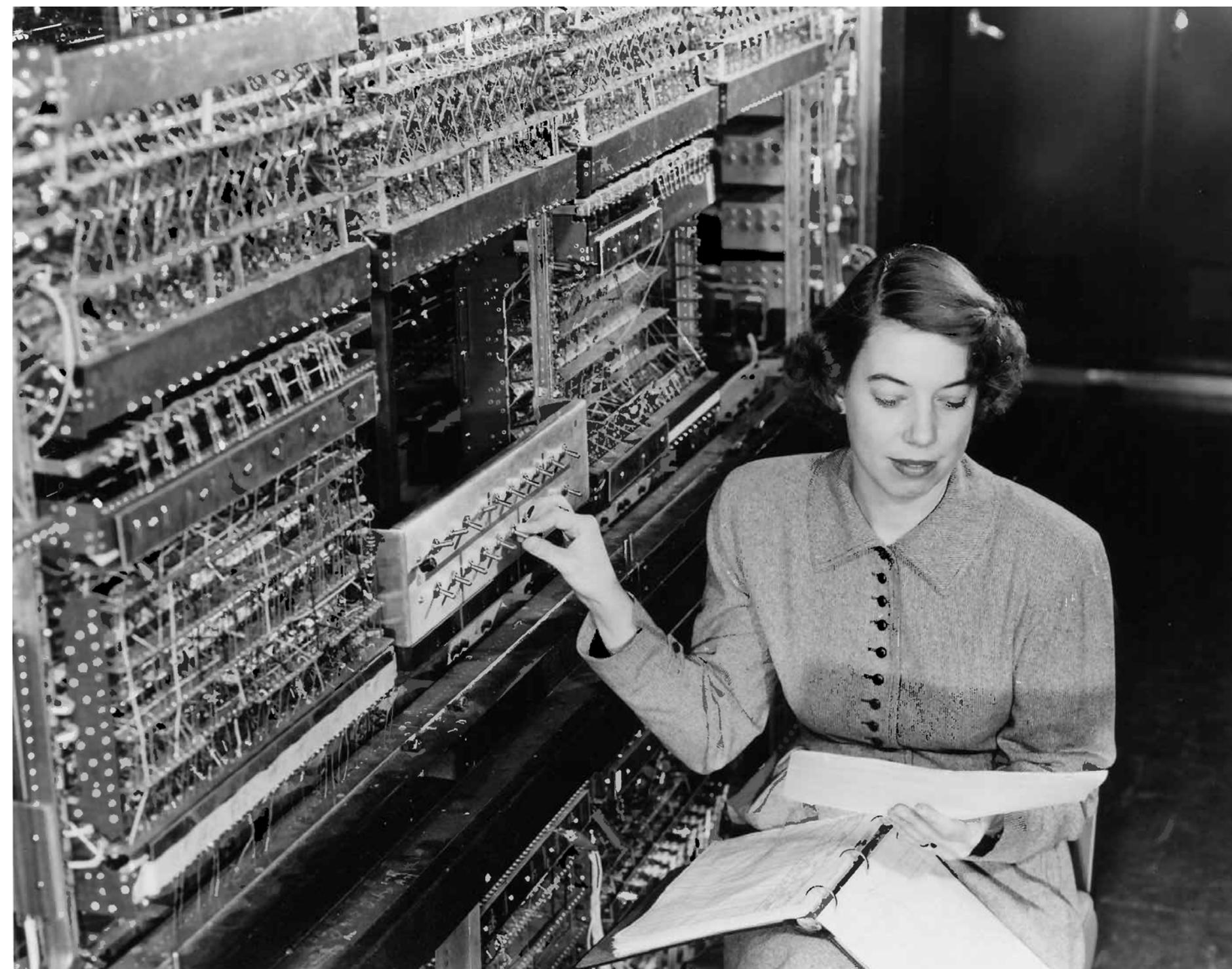




and under
computational
constraints

the ability to reason
in algorithmic terms



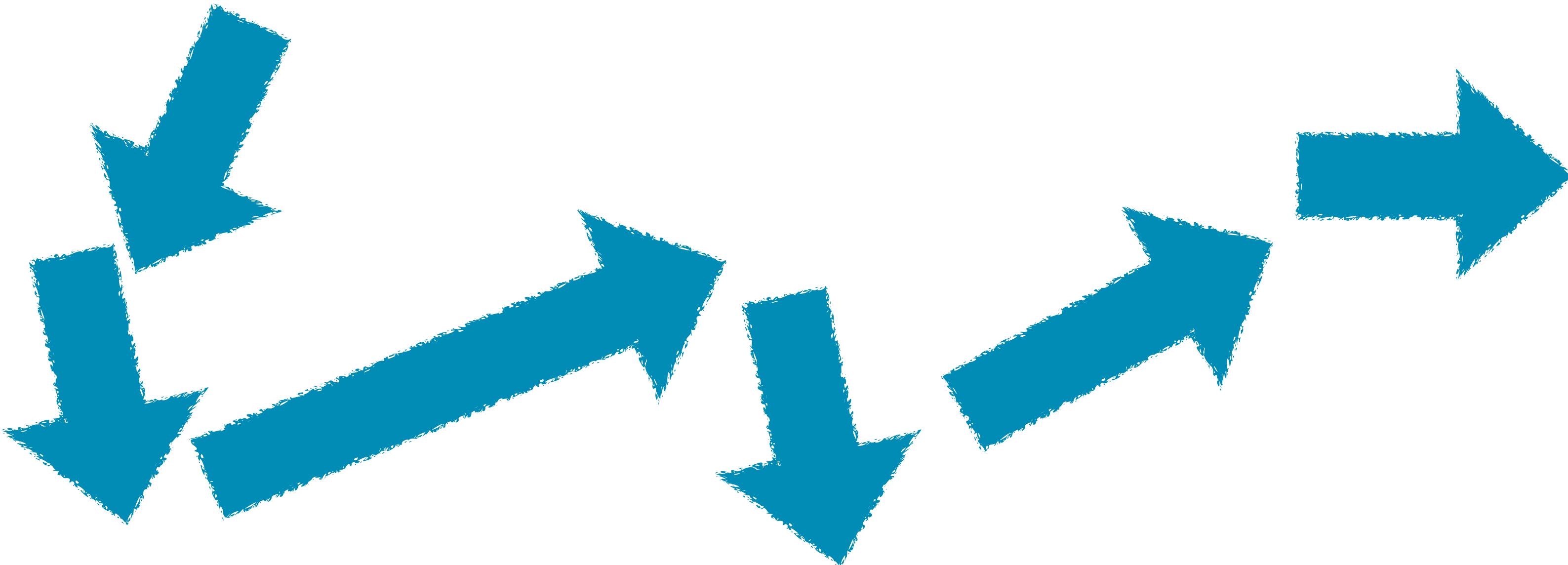


and solve them by writing
computers programs

the ability to precisely
specify various problems



indirect benefits



learn to think differently

learn to think
creatively

experiment
the design attitude

learn to navigate through
different levels of abstraction

be ready for the
digital transformation

decision attitude

assumes that the alternative courses of action are **ready at hand**, including the best one

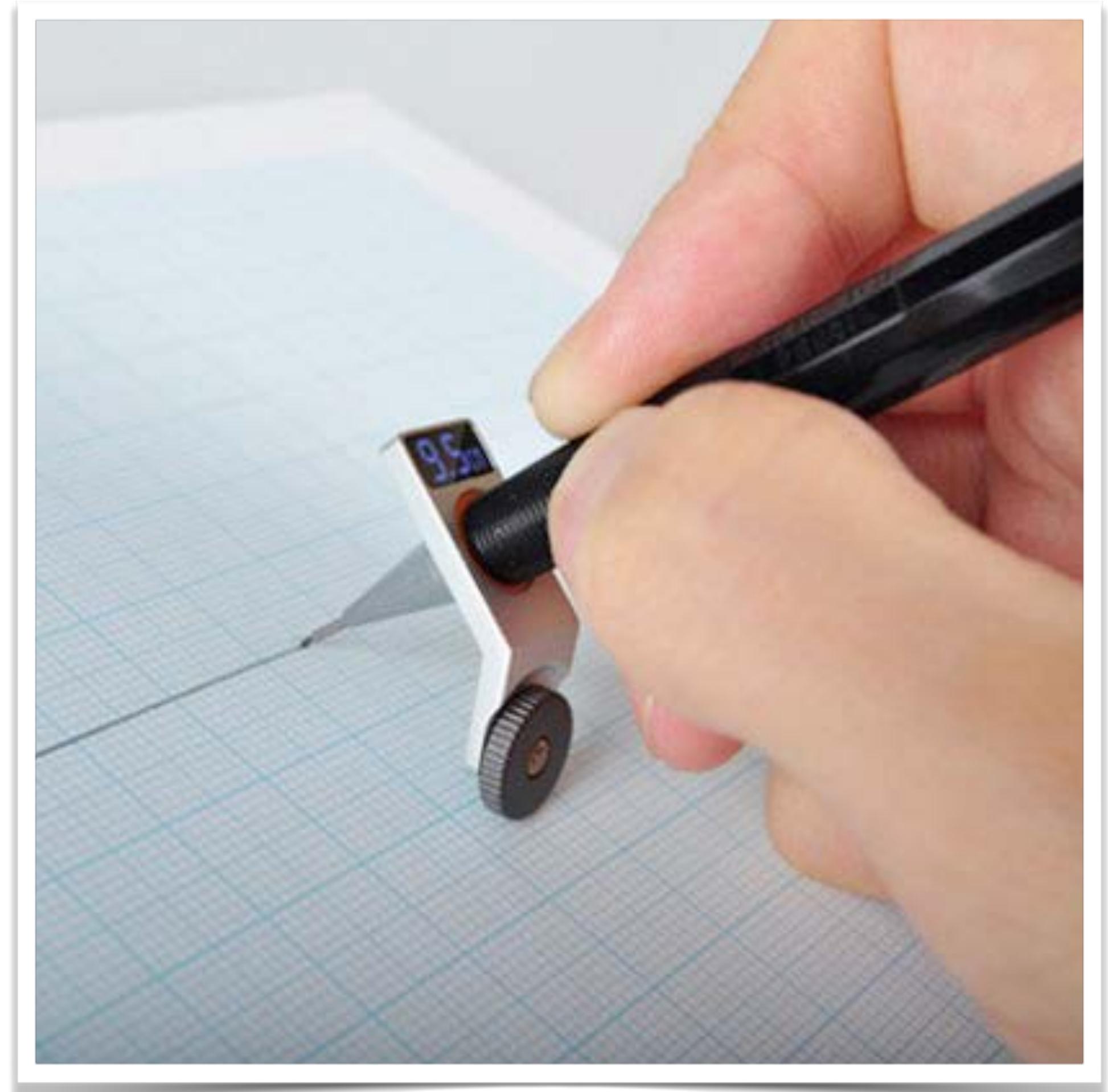
passive view of the decision maker as a problem solver



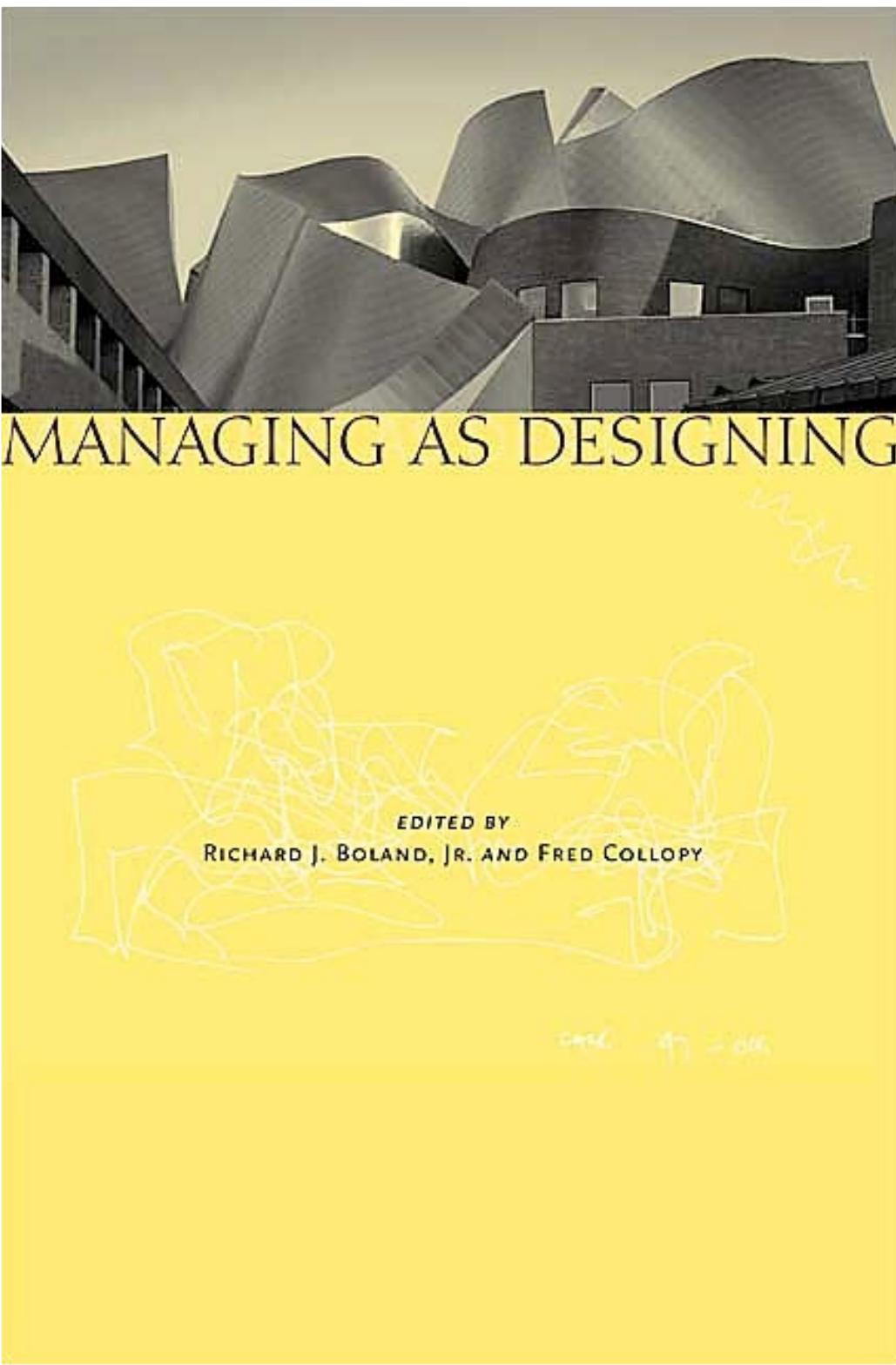
design attitude

a design attitude views each project as an opportunity for invention that includes a questioning of basic assumptions

designers relish the lack of predetermined outcomes



managing as designing

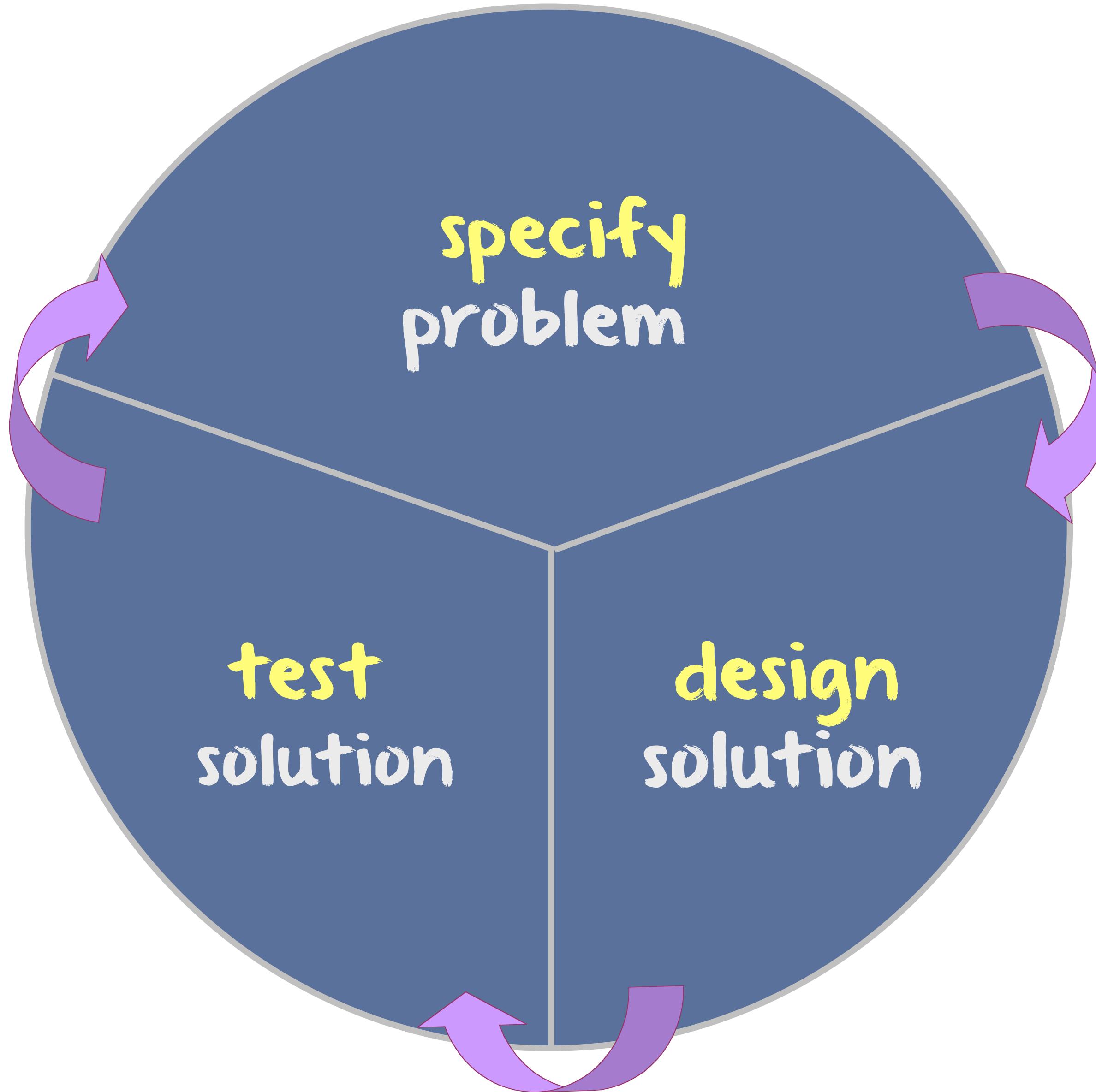


Managing as Designing
R. Boland, F. Collopy
Stanford Press

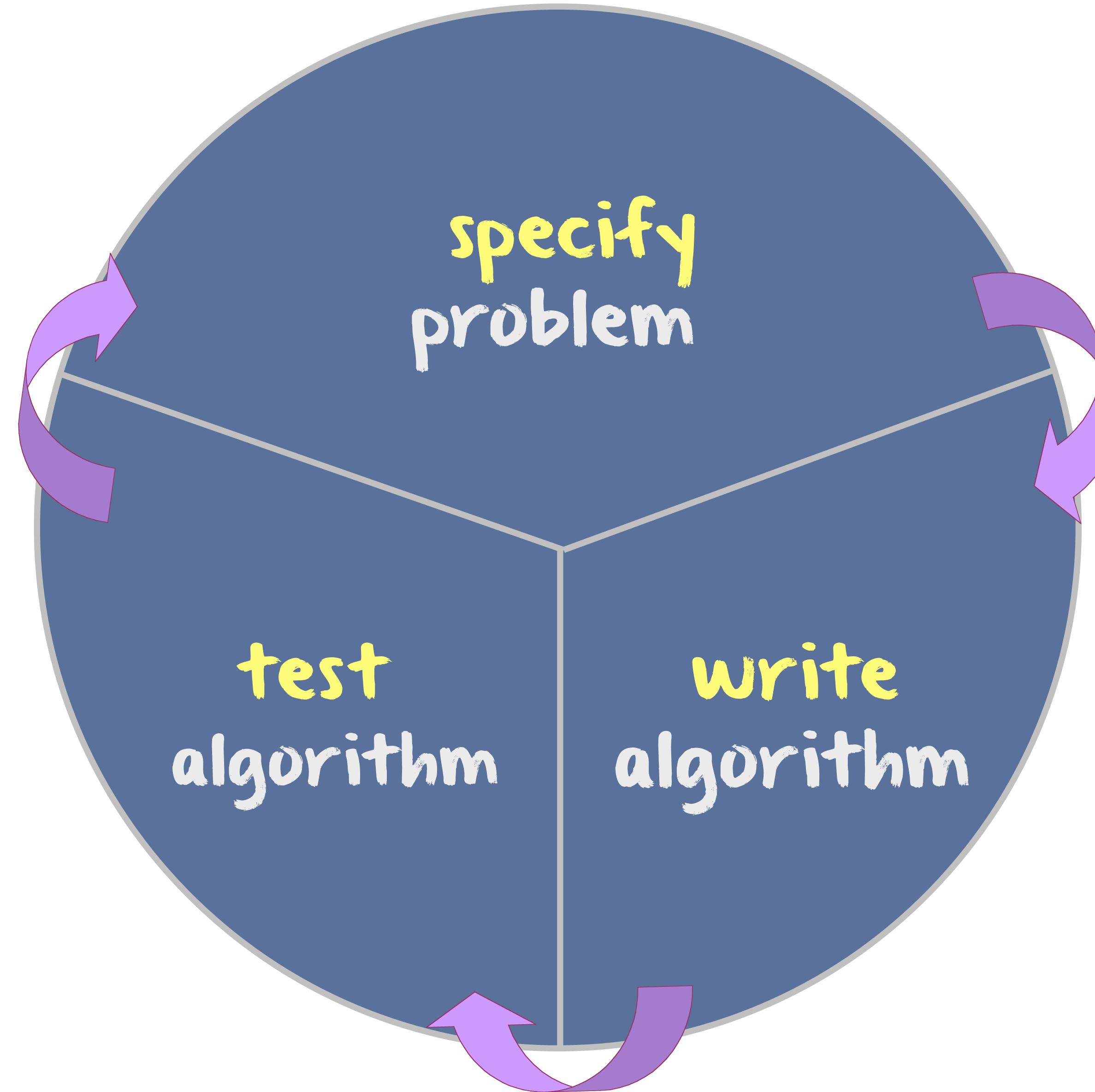
managers should act not
only as decision makers,
but also as designers

though decision and design are
linked in management action,
managers and scholars have
emphasized the decision face over
the design face.

typical design cycle



software development cycle



think abstractions

an abstraction is a set of common properties and laws extracted from several particular examples

examples:

$$\sum \vec{F} = m\vec{a}$$

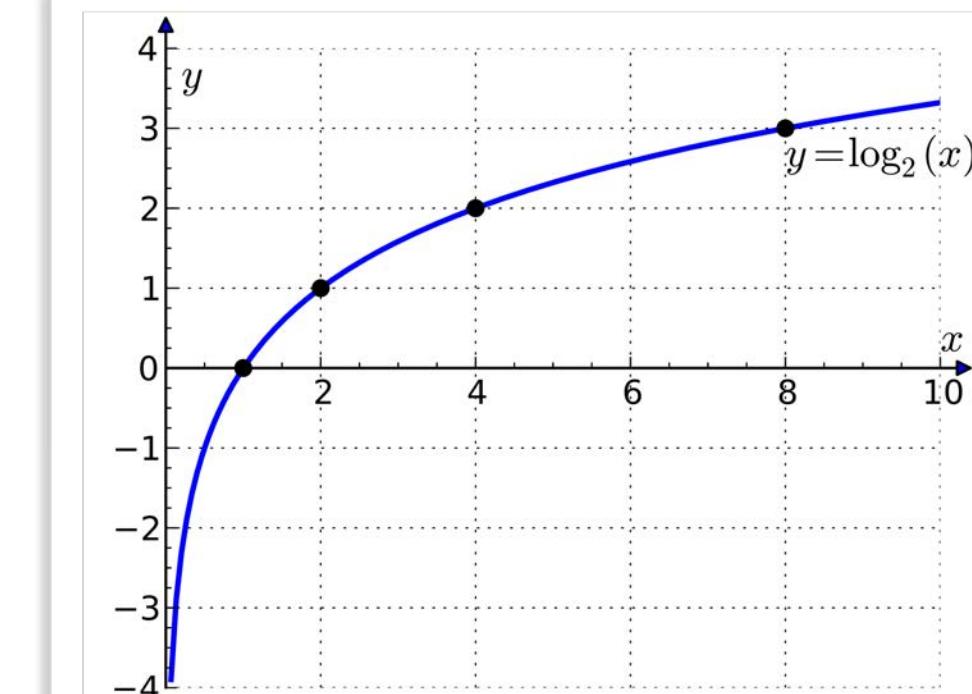
The alteration of motion is ever proportional to the motive force impressed, and is made in the direction of the right line in which that force is impressed

Mutationem motus proportionalem esse vi motrici impressae, et fieri secundum lineam rectam qua vis illa imprimitur

$$f(x, y) = \sqrt{x^2 + y^2}$$

mammals

sphere



thinking in abstractions is one of
the key traits in human being

stacking abstractions

THEOREM 1. $\mathcal{Q} \succeq \mathcal{P}$, $\mathcal{W} \succeq \mathcal{S}$, $\diamond \mathcal{Q} \succeq \diamond \mathcal{P}$, and $\diamond \mathcal{W} \succeq \diamond \mathcal{S}$.

PROOF. Let \mathcal{D} be any failure detector in \mathcal{Q} , \mathcal{W} , $\diamond \mathcal{Q}$, or $\diamond \mathcal{W}$. We show that $T_{\mathcal{D} \rightarrow \mathcal{D}'}$ transforms \mathcal{D} into a failure detector \mathcal{D}' in \mathcal{P} , \mathcal{S} , $\diamond \mathcal{P}$, or $\diamond \mathcal{S}$, respectively. Since \mathcal{D} satisfies weak completeness, by Lemma 1, \mathcal{D}' satisfies strong completeness.

LEMMA 1. $T_{\mathcal{D} \rightarrow \mathcal{D}'}$ satisfies P1.

PROOF. Let p be any process that crashes. Suppose that there is a time t after which some correct process q permanently suspects p in $H_{\mathcal{D}}$. We must show that there is a time after which every correct process suspects p in $output^R$.



algorithms
your software
system software
operating system
hardware

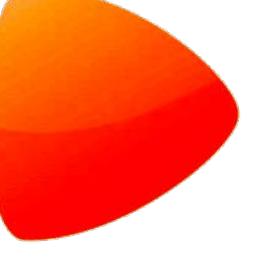
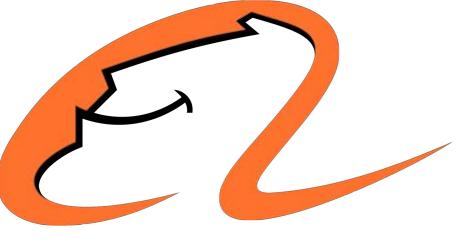
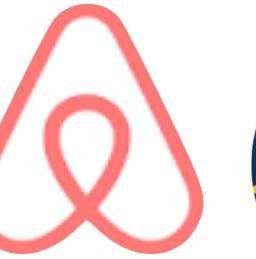
get ready for the digital transformation

software is not longer being simply used as support for existing human activities but rather becoming **the driver of profound changes in the way we do things** and even the source of totally new activities

because for better
or for worse...

... software is now
ruling the world



a  L  GO   LE SHOP.CH   
software is now
ruling the world 
   G  f  N  U  BlaBlaCar 

did you know...



... the largest taxi company owns no vehicles?



... the largest accommodation provider owns no building?



... the most valuable retailer has no inventory?



... the largest travel agency has no public offices?

the amazon effect



the amazon effect



shopping malls: the american dream

the amazon effect



shopping malls:
now zombieland...

books & papers

Viewpoint | Jeannette M. Wing

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 methods and models give us the courage to solve problems and design systems that no one of us would be capable of tackling alone. Computational thinking confronts the riddle of machine intelligence: What can humans do better than computers? and What can computers do better than humans? Most fundamentally it addresses the question: What is computable? Today, we know only parts of the answers to such questions.

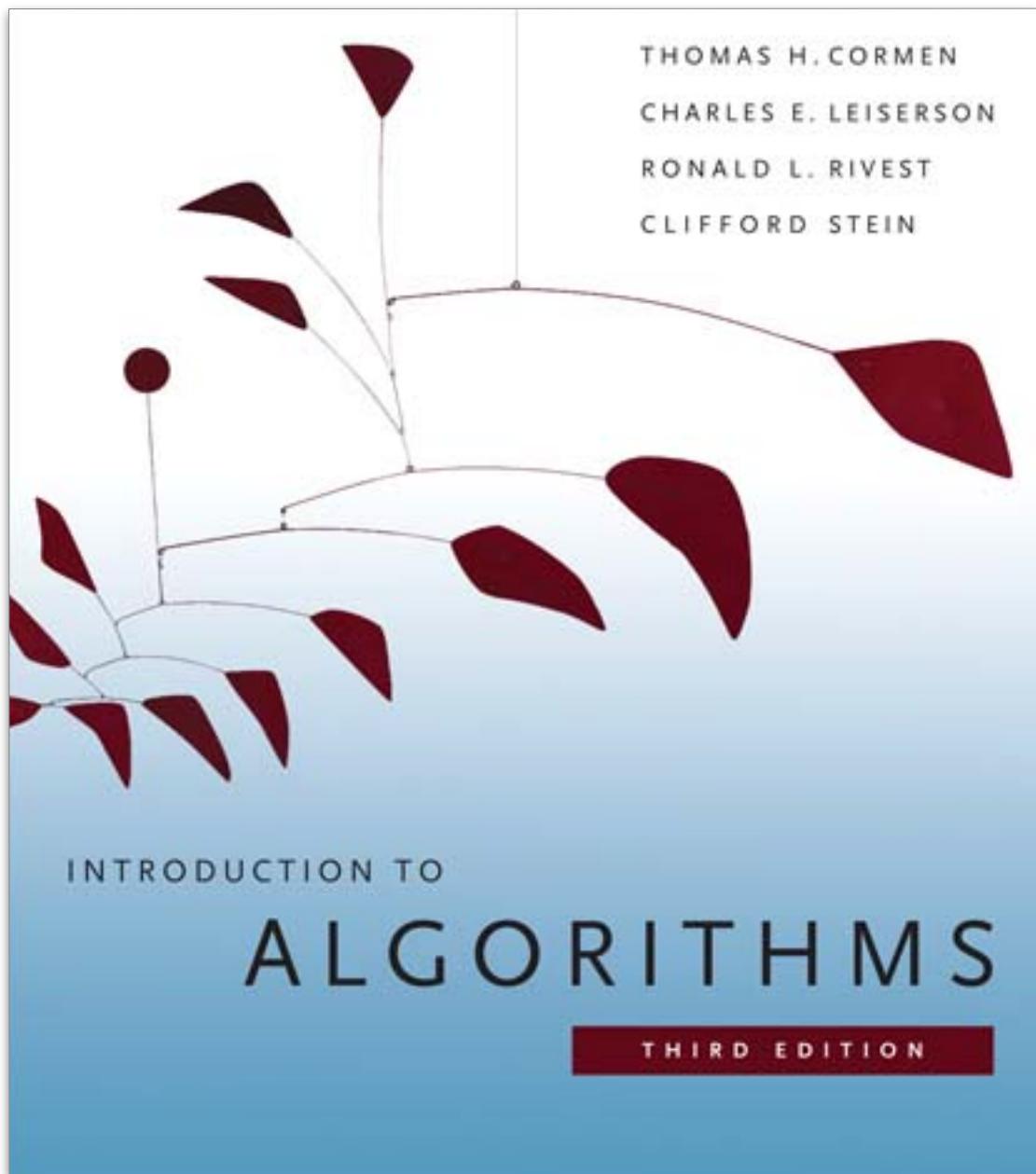
Computational thinking is a fundamental skill for everyone, not just for computer scientists. To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability. Just as the printing press facilitated the spread of the three Rs, what is appropriately incestuous about this vision is that computing and computers facilitate the spread of computational thinking.

Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science. Computational thinking includes a range of mental tools that reflect the breadth of the field of computer science.

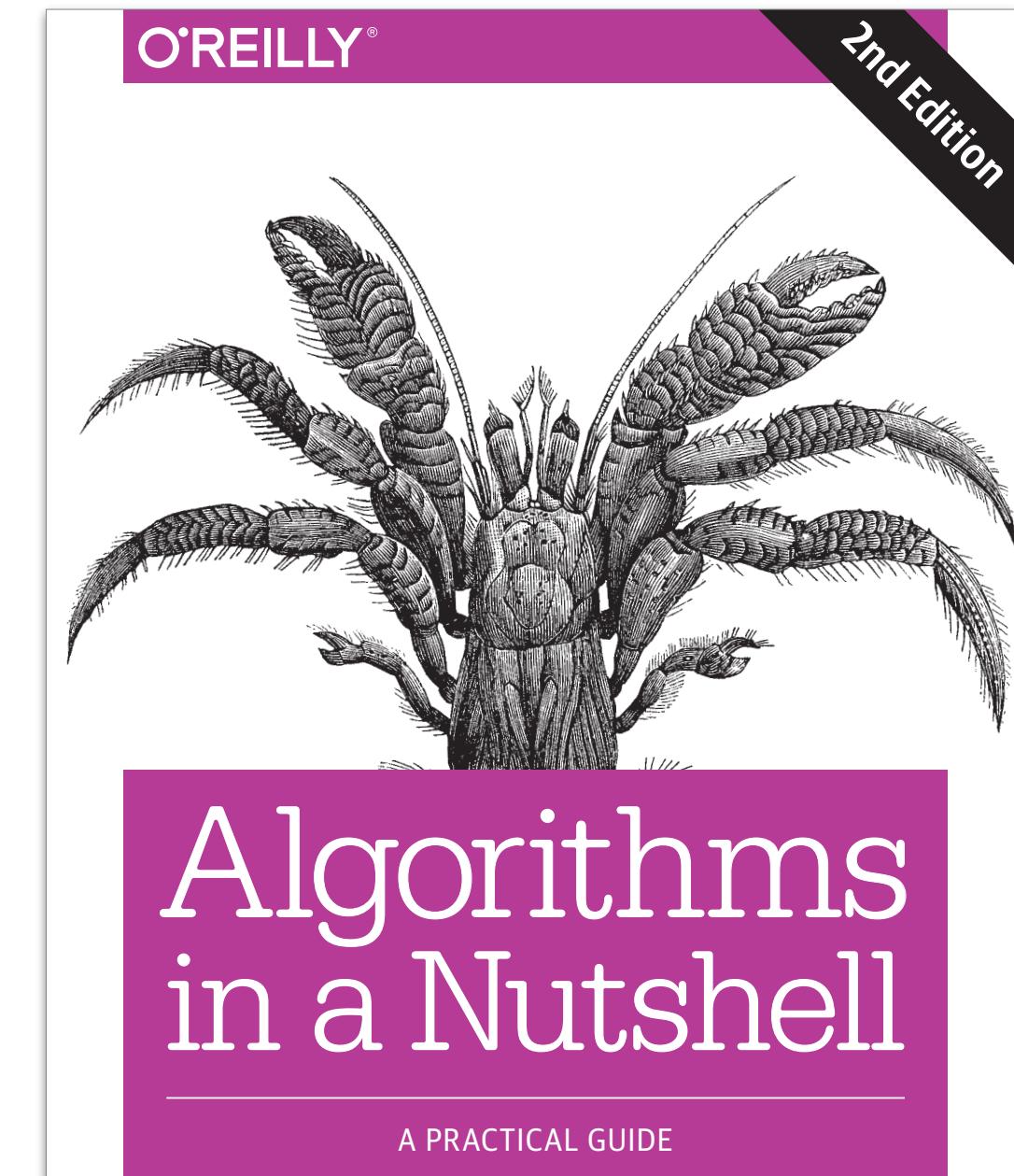
Having to solve a particular problem, we might ask: How difficult is it to solve? and What's the best way to solve it? Computer science rests on solid theoretical underpinnings to answer such questions pre-

COMMUNICATIONS OF THE ACM March 2006/Vol. 49, No. 3 33

Computational thinking. J.M. Wing.
Communication of the ACM,
49(3):33–35, March 2006.



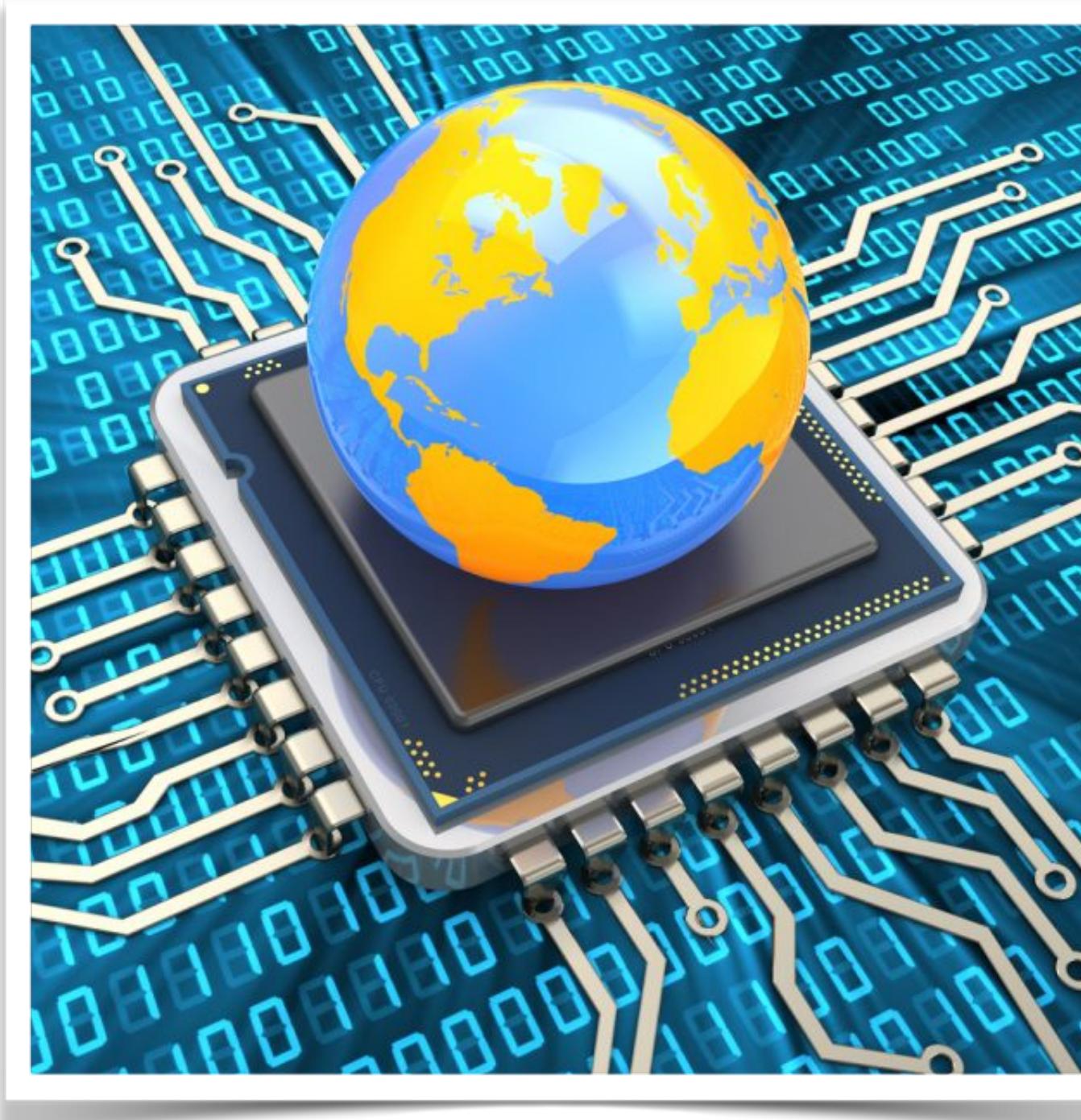
Introduction to Algorithms, 3rd Edition.
T.H.Cormen, C.E. Leiserson, R.L. Rivest,
C. Stein. July 2009. MIT Press.



**Algorithms
in a Nutshell**
A PRACTICAL GUIDE

Algorithms in a Nutshell, 2nd Edition.
G.T. Heineman, G. Pollice, S. Selkow.
March 2016. O'Reilly.

overview - week 1



computer
architecture

overview - week 2



system
software

overview - week 3



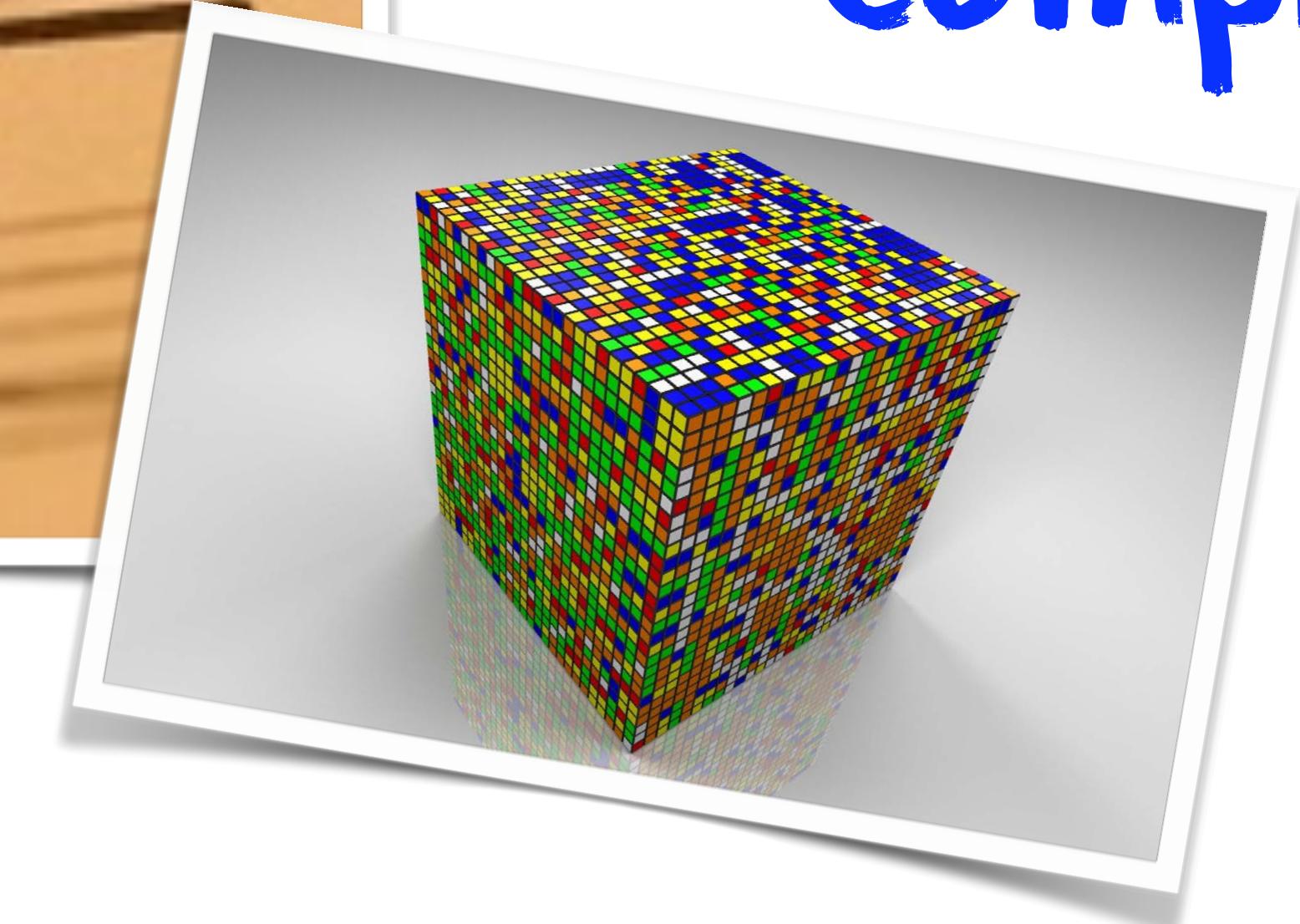
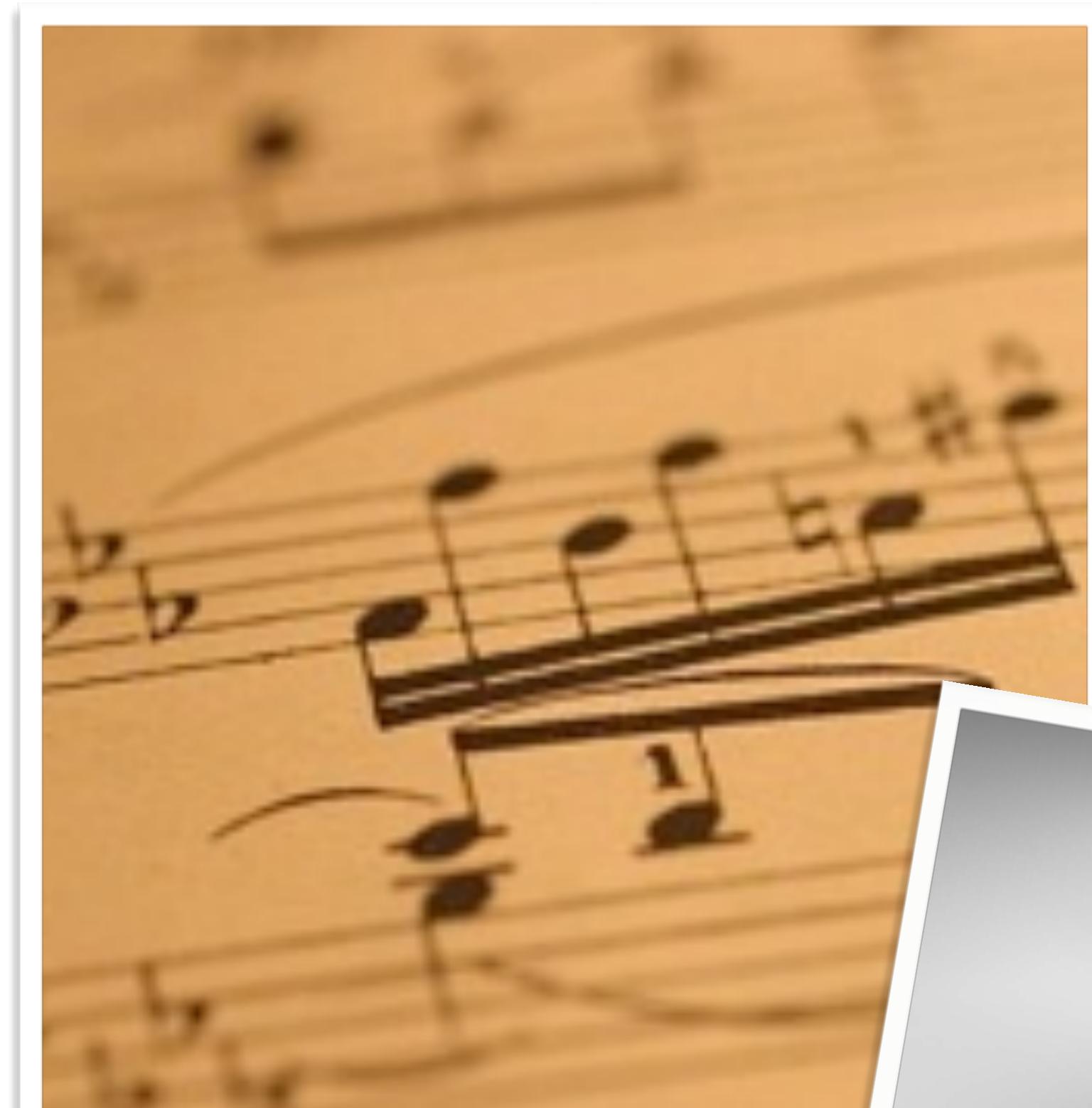
programming
basics

overview - week 4



induction &
recursion

overview - week 5



algorithms &
computational
complexity

overview - week 6



searching
algorithms

overview - week 7



graph
algorithms

overview - week 8

mid-term test



overview - week 9

mid-term test
correction



overview - week 10



spatial tree
algorithms

overview - week II



probabilistic
algorithms

overview - week 12



classes,
objects &
methods

overview - week 13



inheritance &
polymorphism

overview - week 14



abstract
classes &
types

Calendar

TUESDAY	8.00-10.00	Group A : 14.15-16.00 + Group B 16.15-18.00
Sep 15	course overview	computer architecture
Sep 22	system software	exercises on computer architecture
Sep 29	programming basics	exercises on system software
Oct 06	induction and recursion	exercises on programming basics
Oct 13	algorithms and their complexity	exercises on induction and recursion
Oct 20	searching algorithms	exercises on algorithms and their complexity
Oct 27	graphs algorithms	exercises on searching algorithms
Nov 03	intermediate test	consolidation exercises
Nov 10	intermediate test – Correction	exercises on graph algorithms
Nov 17	spatial tree algorithms	consolidation exercises
Nov 24	probabilistic algorithms	exercises on spatial tree algorithms
Dec 01	classes, objects and interfaces	exercises on probabilistic algorithms
Dec 08	inheritance and polymorphism	exercises on classes, objects and interfaces
Dec 15	abstract classes & types	exercises on inheritance and polymorphism
		exercises on abstract classes & types

Legend:



this exercise session is moved
to Friday 6th November



programming languages



python

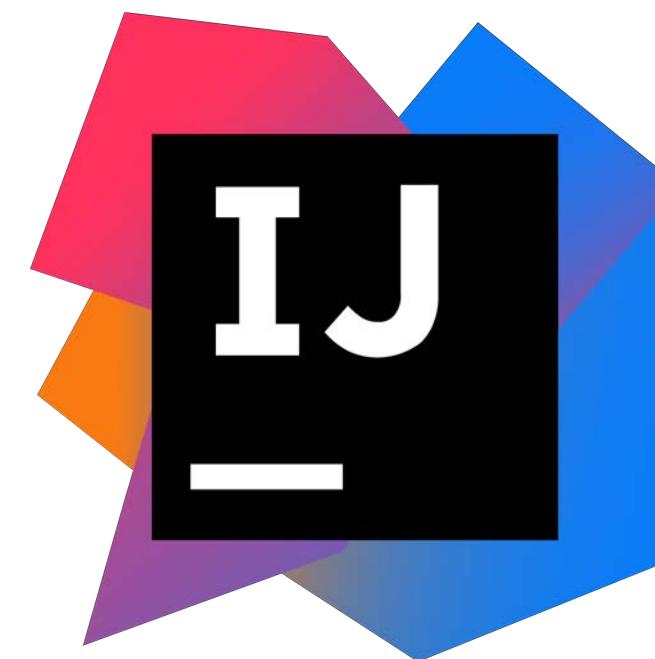


java

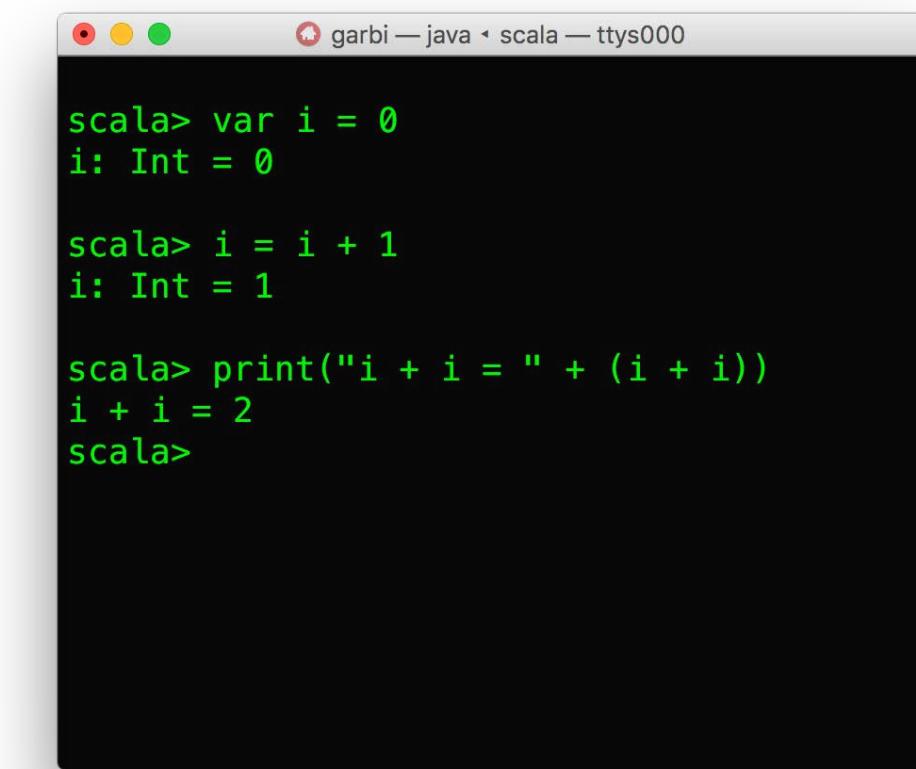
development tools



PyCharm
(python)



IntelliJ
(python / java)

A screenshot of a Mac OS X terminal window titled 'garbi — java • scala — ttys000'. The window displays the following Scala code:

```
scala> var i = 0
i: Int = 0

scala> i = i + 1
i: Int = 1

scala> print("i + i = " + (i + i))
i + i = 2
scala>
```

the good old
terminal



practical issues each week



1/3 of the students follow the course
in the class room

2/3 of the students follow the course
online via streaming

lectures: Amphimax 351 ←
exercises: Amphipôle 140 + 146 ←

- you will be dispatched into two groups
- support will only be provided online

practical issues

web sites

doplab.unil.ch/act

doplab.unil.ch/act-moodle



evaluation

the evaluation is based on

- an intermediate **test** during the semester
- a final **exam*** during the exam session



$$\text{grade} = 0.4 \times \text{test} + 0.6 \times \text{exam}$$

* the final exam is written in the regular session
and either oral or written in the retake session