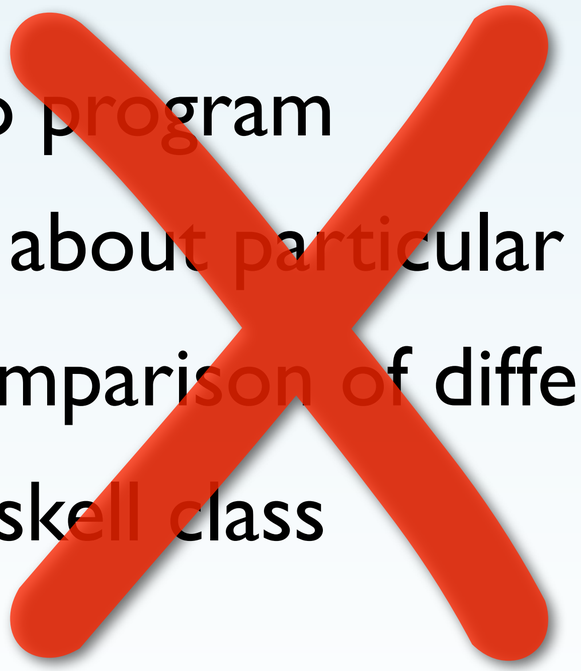
Three Expectations about CS 381

- I learn how to program
- I learn details about particular programming languages
- I will see a comparison of different programming languages
- This is the Haskell class

Three Myths about CS 381



- I learn how to program
- I learn details about particular programming languages
- I will see a comparison of different programming languages
- This is the Haskell class

What is CS 381 about?

*What **exactly** is a programming language?*

Recursion?
❺ *Runtime Stack*

```
int x;
int f(int z) {x := f(y) …};
{ int x;
  f(3)
}
```

Value or address?
When to evaluate?
❹ *Parameter Passing*

```
x := f(y) + g(z+1)
```

Which x?
❷ *Scoping*

"Determine" value of RHS and "assign it" to `x`

❾ *Semantics*

Based on what?
❶ *Abstract Syntax*

Imperative programming =
Programming "by sticky notes"

Alternatives?
❽ *Programming Paradigms*

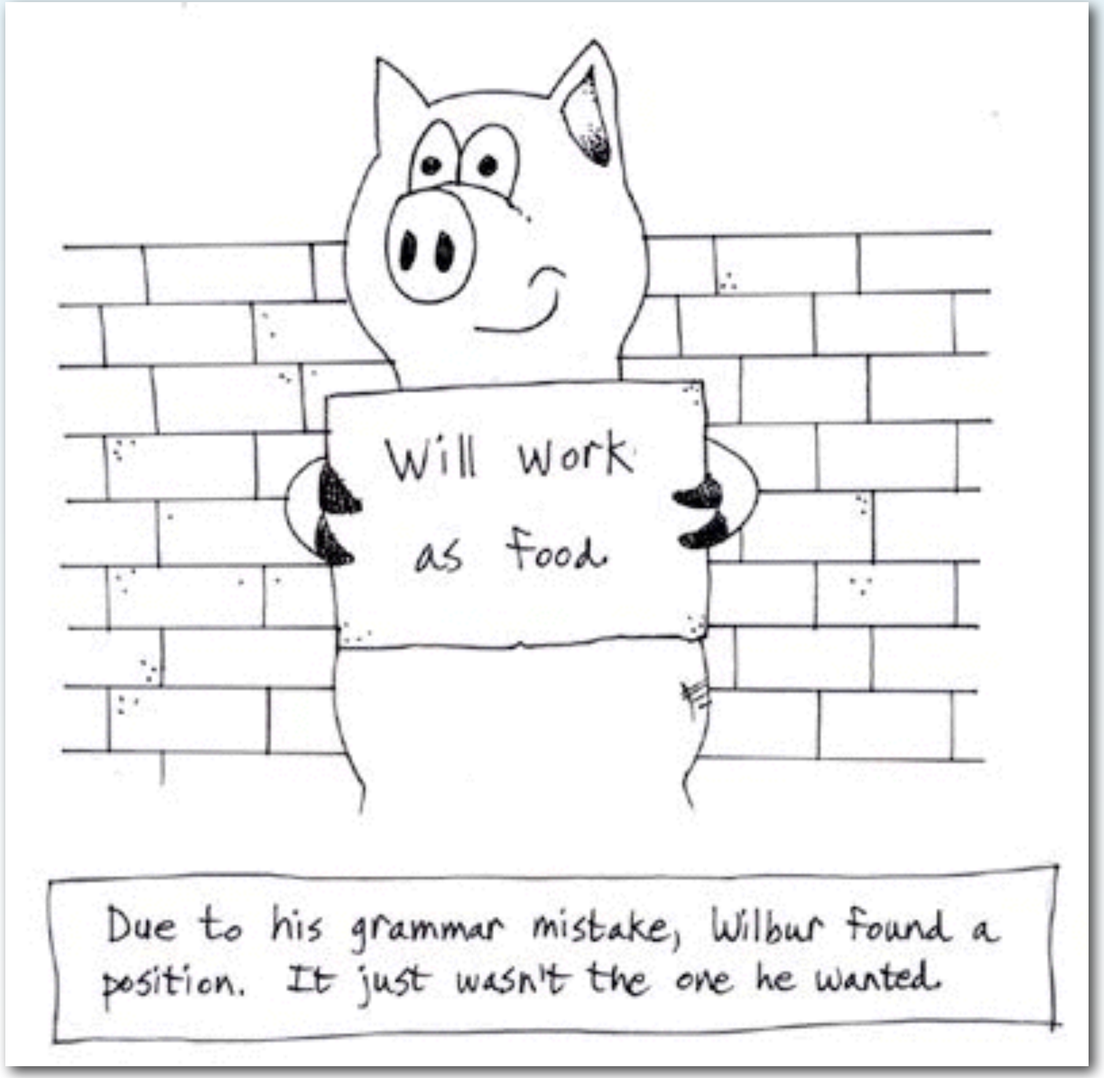
The Role of Haskell in CS 381

- *Example* of a non-imperative programming paradigm (just like Prolog)
- *Tool* for describing language concepts (syntax, semantics, scope, typing)

	Metalanguage		
	English	Math	Haskell
Precise	✗	✓	✓
Checkable	✗	✗	✓
Executable	✗	✗	✓

1 Introduction

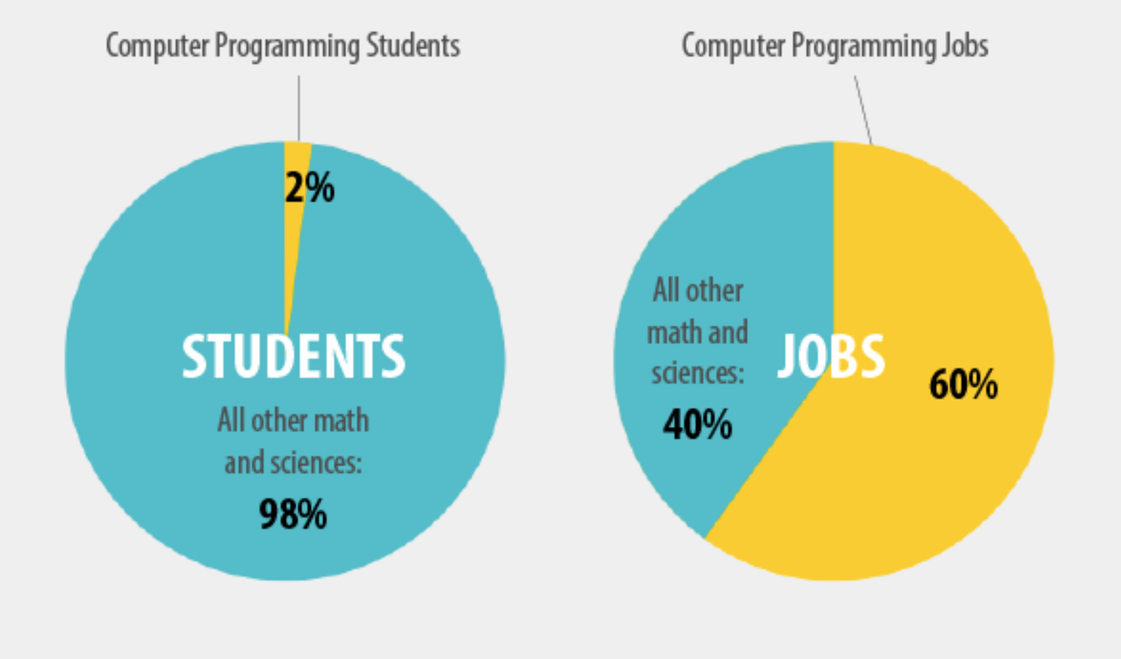
The correct use of language can be critical



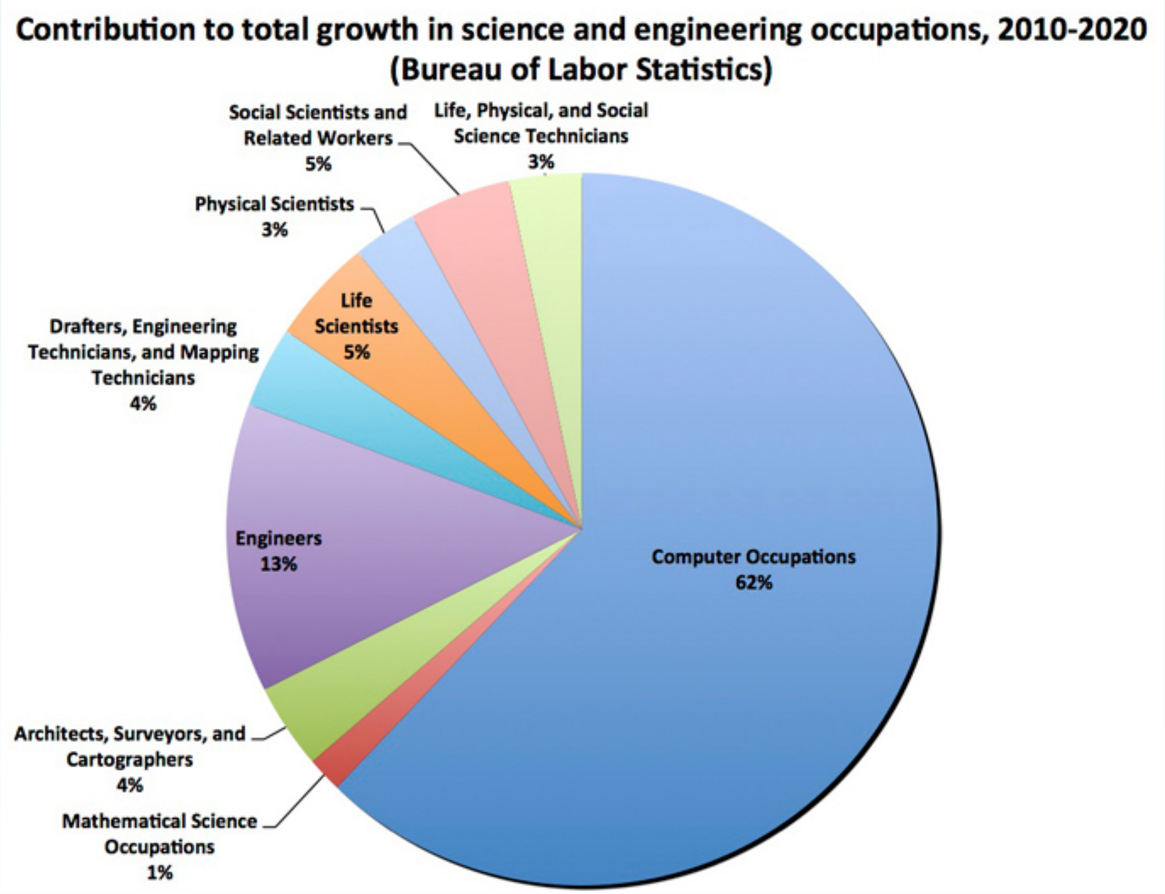
1 Introduction

- About "computer science"
- The role of programming languages
- How to study programming languages?

CS Students & CS Profession



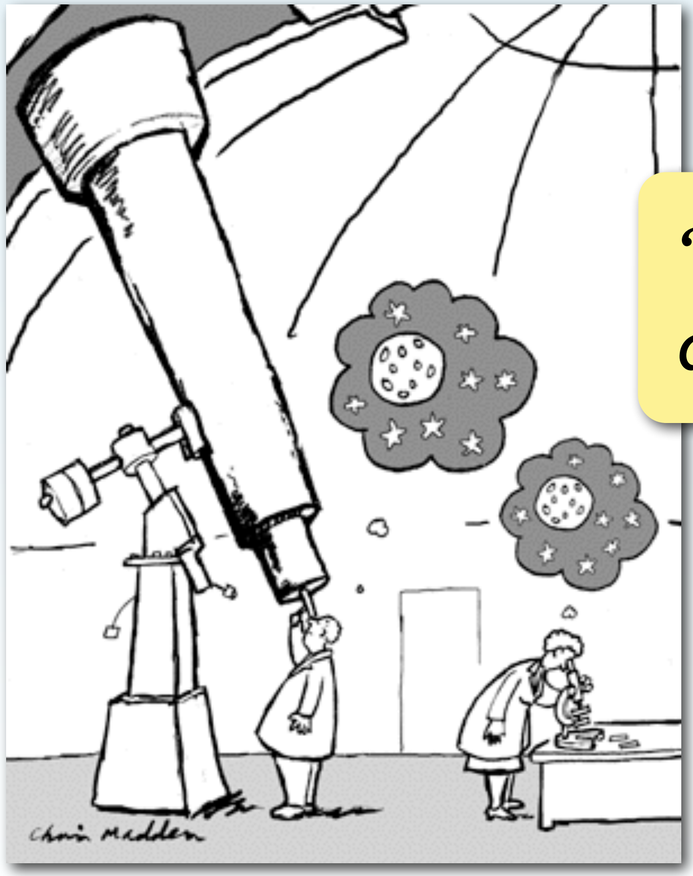
LESS THAN 2.4% OF COLLEGE STUDENTS GRADUATE WITH A DEGREE IN COMPUTER SCIENCE... THAT'S FEWER THAN 10 YEARS AGO.



CS Stereotypes

Countess Ada Lovelace

What is Computer Science?

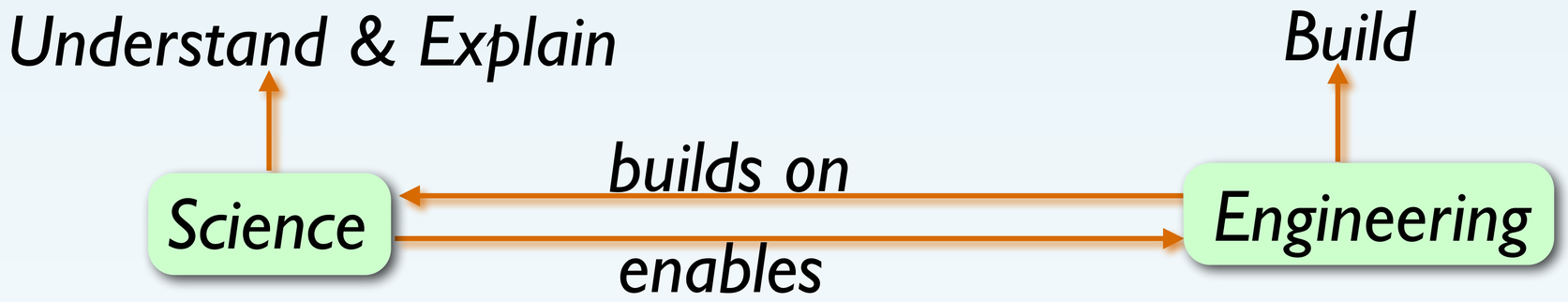


"Computer science is no more about computers than astronomy is about telescopes."

Edsger Dijkstra (?)

Computer Science = The Science of Computing

Science and Engineering



Physics
Chemistry
Theoretical CS

ME, EE, ...
CE, ...
SE, ...

What is Computation?

Systematic Transformation of Representation

Systematic
Intensional
Description

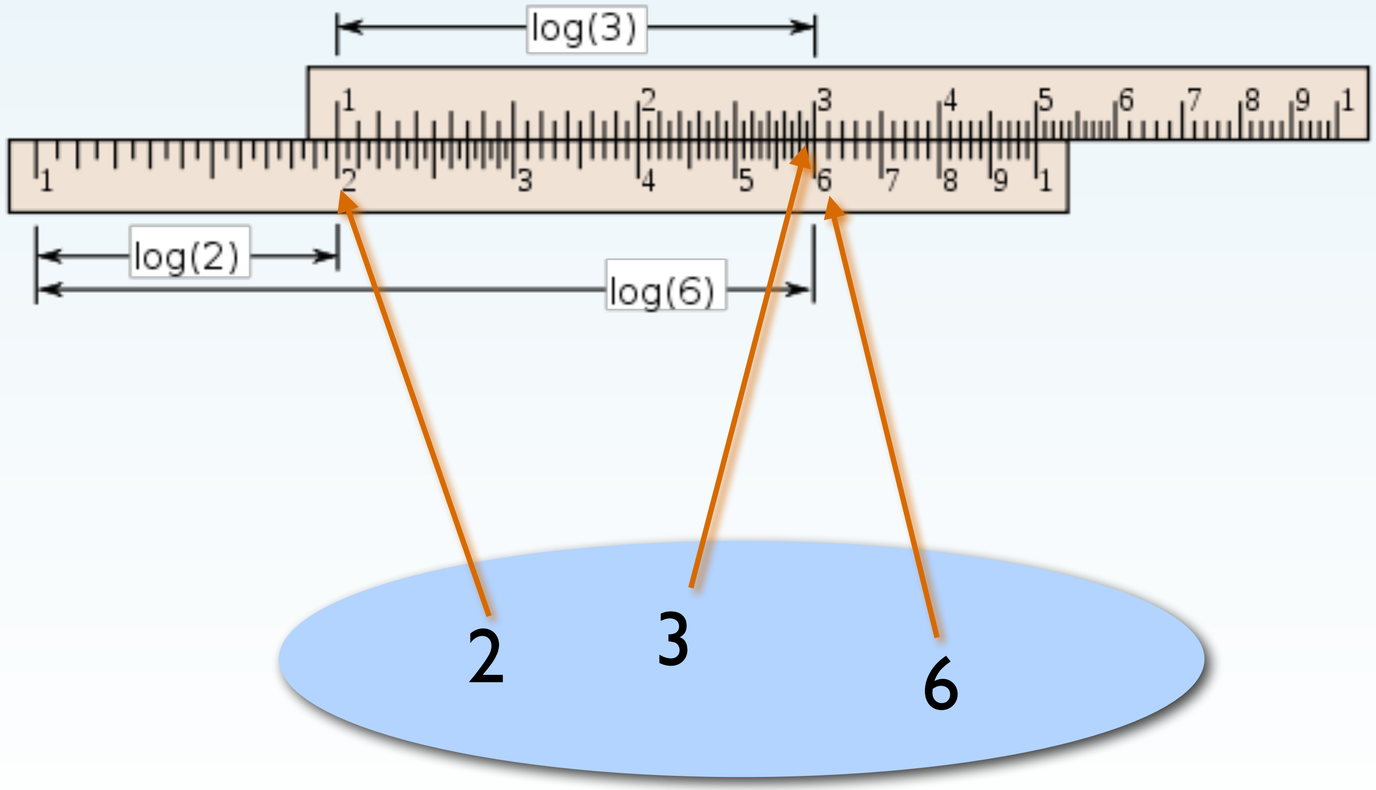
Transformation
Function

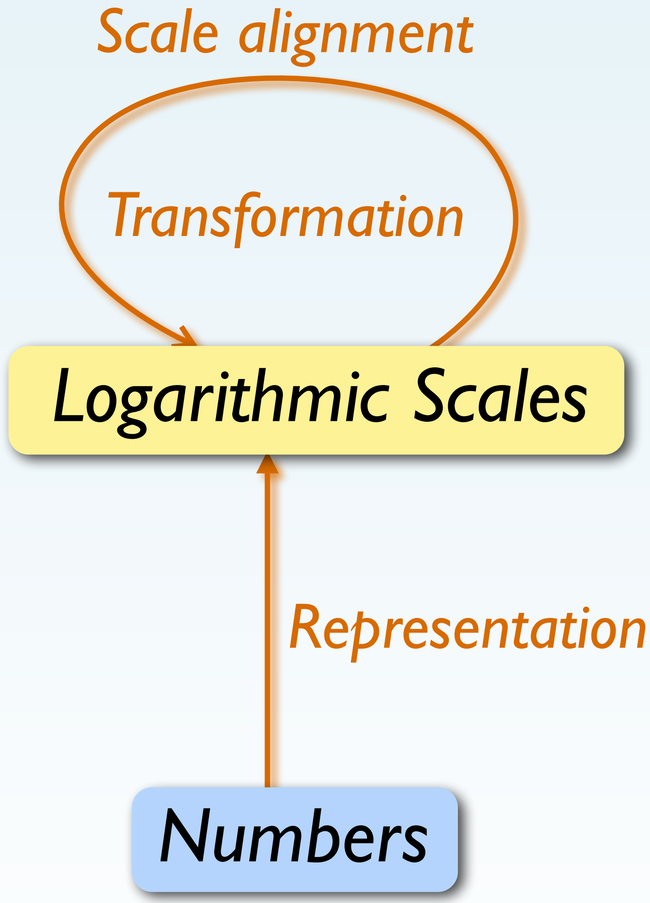
Representation
Abstraction that preserves particular features

Description given in a Language

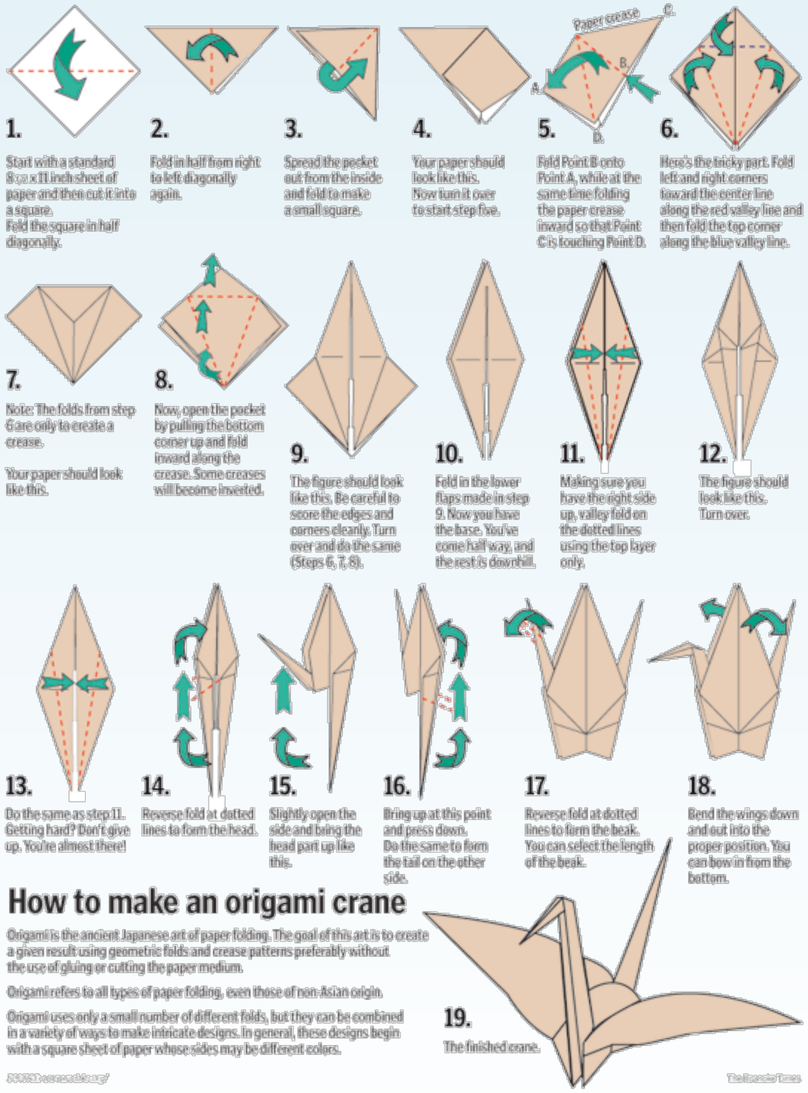
Multiplication

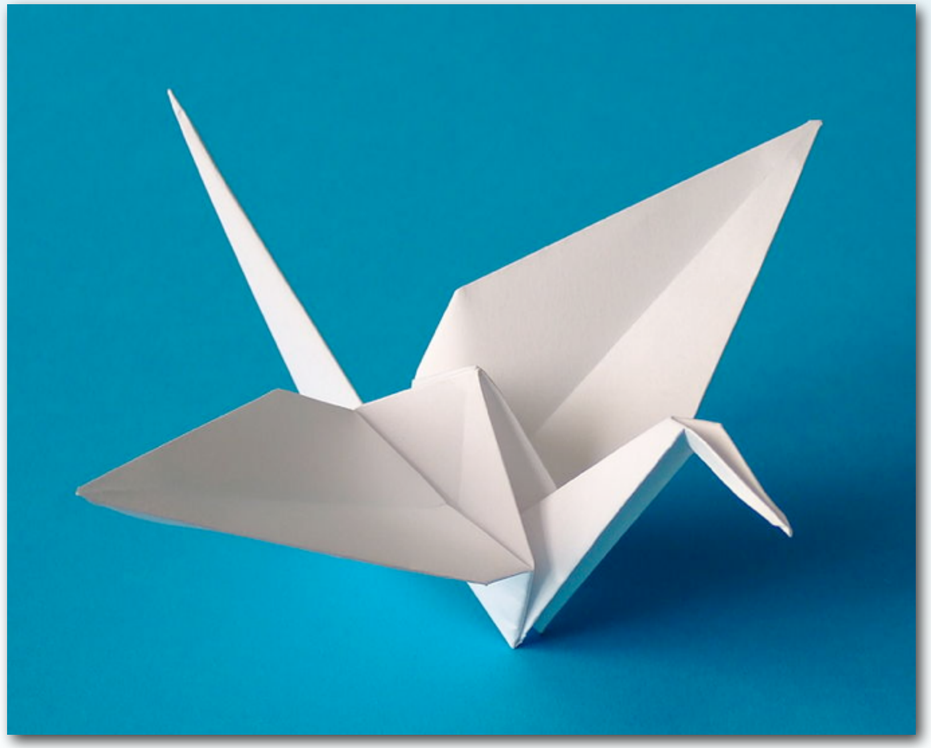
$$\log (x \cdot y) = \log x + \log y$$

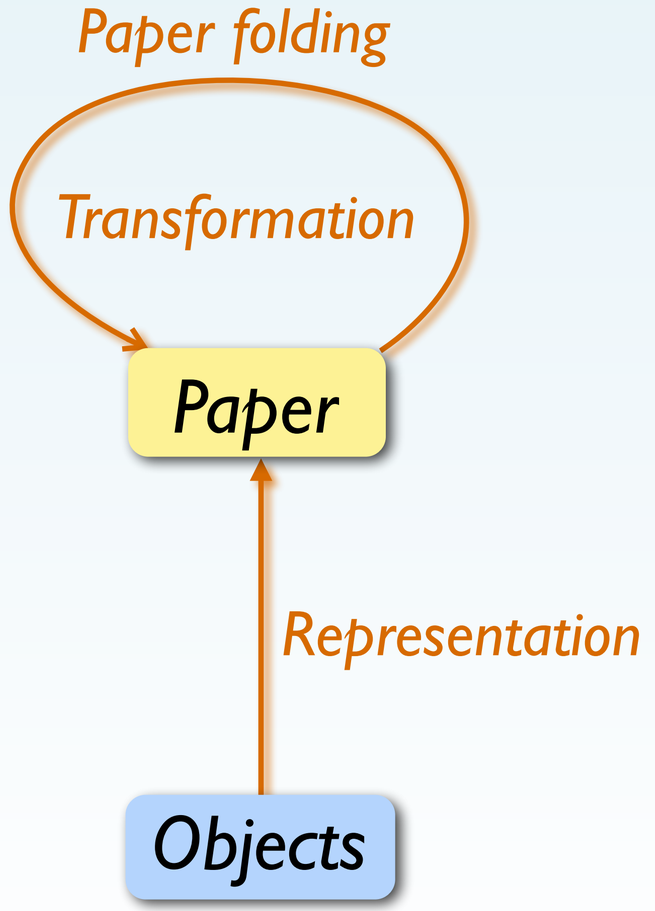




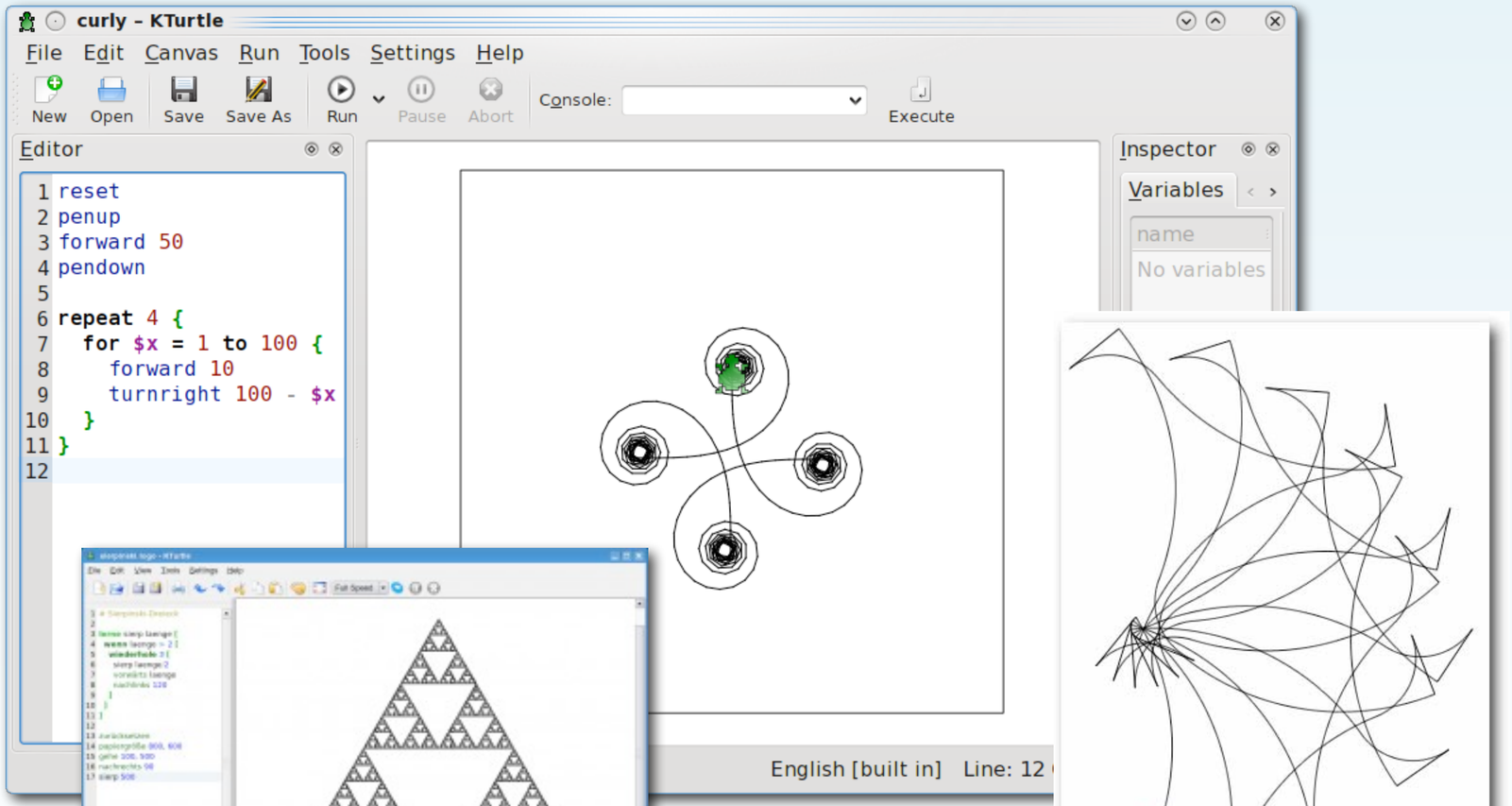
Origami


Paper crease
A
B
C
D
1.
Start with a standard 8 1/2 x 11 inch sheet of paper and then cut it into a square.
Fold the square in half diagonally.
2.
Fold in half from right to left diagonally again.
3.
Spread the pocket out from the inside and fold to make a small square.
4.
Your paper should look like this.
Now turn it over to start step five.
5.
Fold Point B onto Point A, while at the same time folding the paper crease inward so that Point C is touching Point D.
6.
Here's the tricky part. Fold left and right corners toward the center line along the red valley line and then fold the top corner along the blue valley line.
7.
Note: The folds from step 6 are only to create a crease.
Your paper should look like this.
8.
Now, open the pocket by pulling the bottom corner up and fold inward along the crease. Some creases will become inverted.
9.
The figure should look like this. Be careful to score the edges and corners cleanly. Turn over and do the same (Steps 6, 7, 8).
10.
Fold in the lower flaps made in step 9. Now you have the base. You've come half way, and the rest is downhill.
11.
Making sure you have the right side up, valley fold on the dotted lines using the top layer only.
12.
The figure should look like this. Turn over.
13.
Do the same as step 11. Getting hard? Don't give up. You're almost there!
14.
Reverse fold at dotted lines to form the head.
15.
Slightly open the side and bring the head part up like this.
16.
Bring up at this point and press down. Do the same to form the tail on the other side.
17.
Reverse fold at dotted lines to form the beak. You can select the length of the beak.
18.
Bend the wings down and out into the proper position. You can bow in from the bottom.
How to make an origami crane
Origami is the ancient Japanese art of paper folding. The goal of this art is to create a given result using geometric folds and crease patterns preferably without the use of gluing or cutting the paper medium.
Origami refers to all types of paper folding, even those of non-Asian origin.
Origami uses only a small number of different folds, but they can be combined in a variety of ways to make intricate designs. In general, these designs begin with a square sheet of paper whose sides may be different colors.
19.
The finished crane.




Paper folding
Transformation
Paper
Representation
Objects

Logo



Moving a "Turtle"

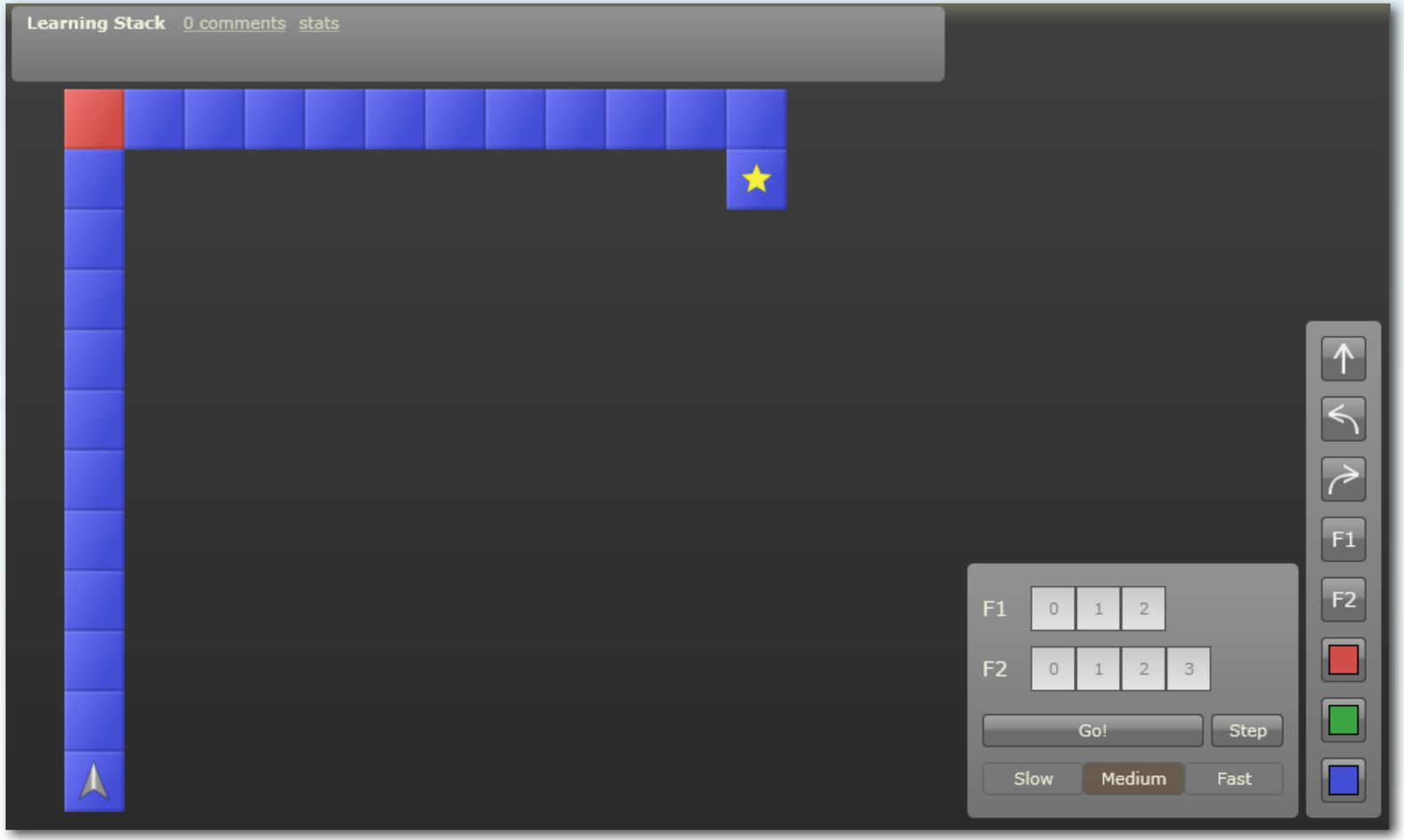
Transformation

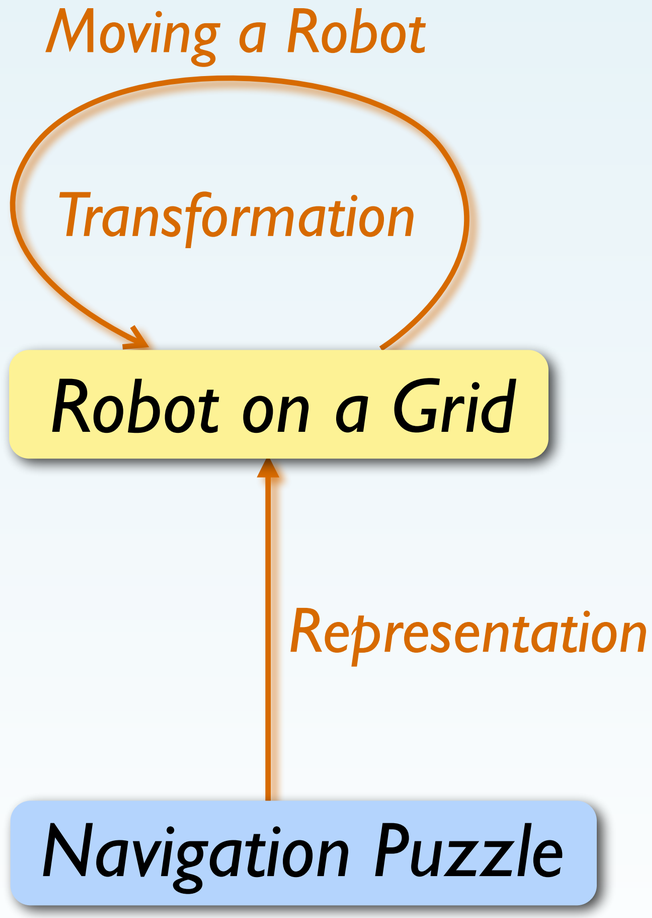
2D Graphics

Representation

Anything visualizable

Robozzle


Learning Stack
0 comments
stats
F1
0
1
2
F2
0
1
2
3
Go!
Step
Slow
Medium
Fast
F1
F2


Moving a Robot
Transformation
Robot on a Grid
Representation
Navigation Puzzle

1 Introduction

- About "computer science"
- The role of programming languages
- How to study programming languages?

Two Aspects

Computer Science

Problem solving

Algorithmic Thinking

"How to do ..."

Design Thinking

"How to represent ..."

HCI
Representations (for problem solving)
Language design
...

Algorithms & Design

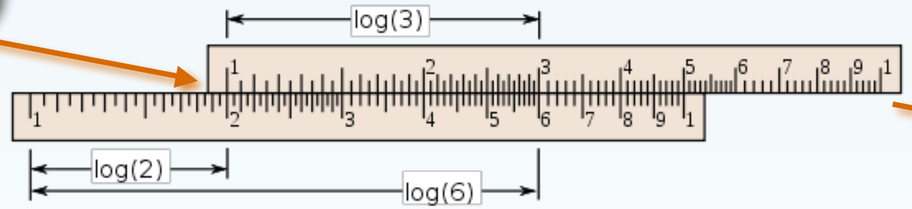
Design Thinking

Algorithmic Thinking

Domain:

Numbers

101100



Rules for aligning scales

By example

English

Visual Rules

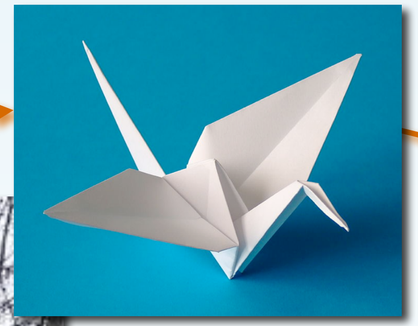
Algorithms & Design

Design Thinking

Algorithmic Thinking

Domain:

Art



Rules for paper folding

By example

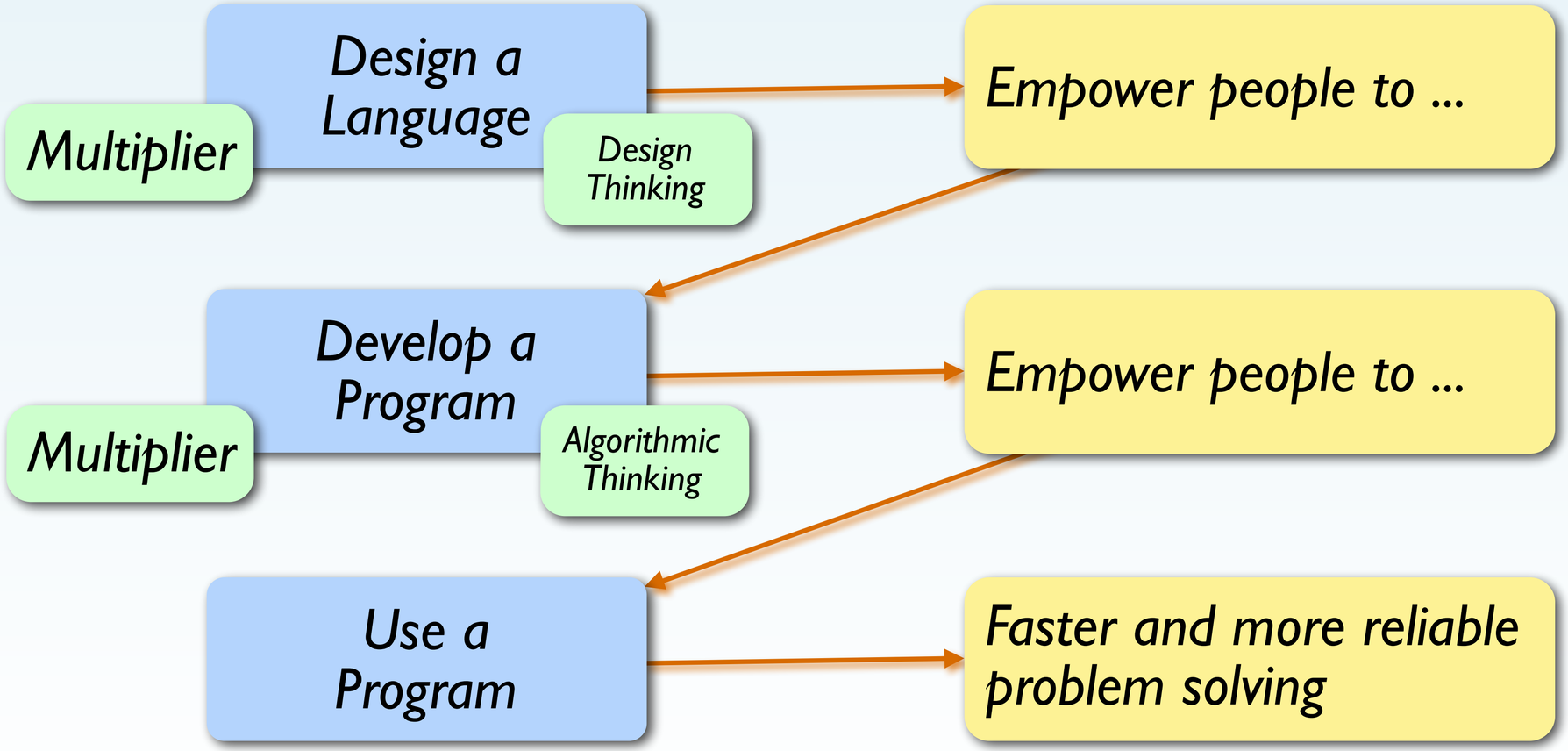
English

Visual Rules

en.wikipedia.org/wiki/Euric

en.wikipedia.org/wiki/Code_of_Euric

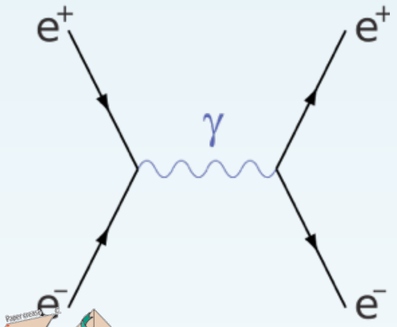
Impact of Languages & Programs

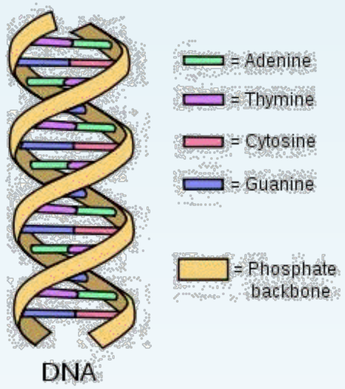


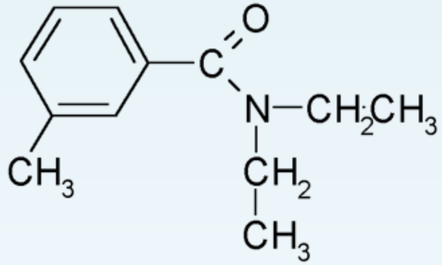
Domain-Specific Languages

$F = ma$

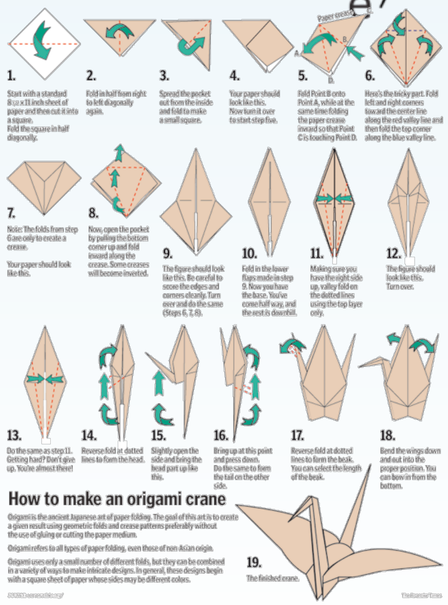
$E = mc^2$

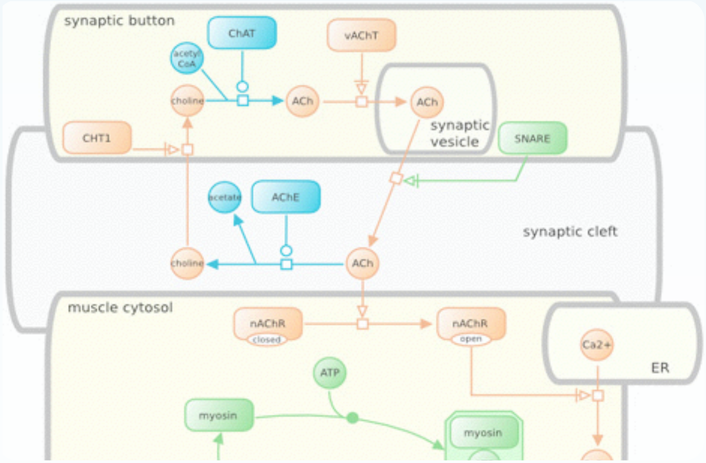


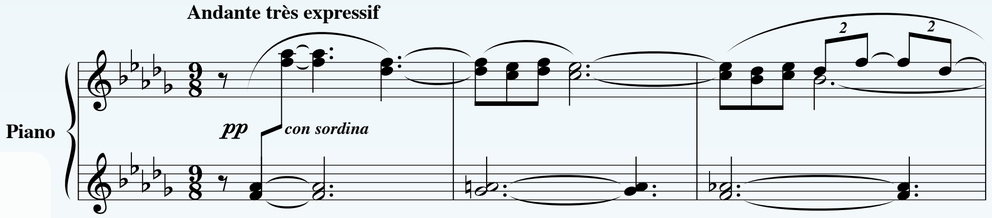


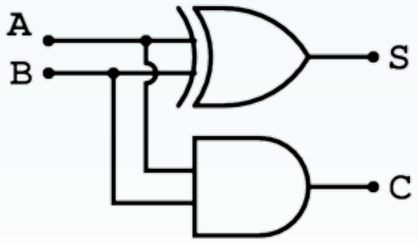


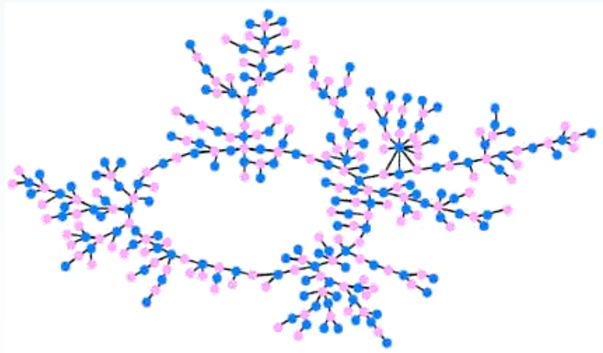
$2H_2 + O_2 \rightarrow 2H_2O$





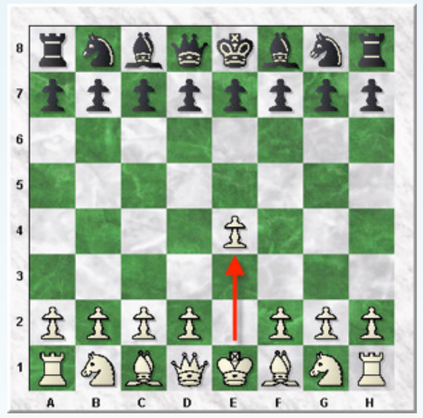






Example Domain: Chess

Many Notations ...



Descriptive
1. P-K4 P-K4
2. N-KB3 N-QB3
3. B-N5 P-QR3
4. BxN QPxB

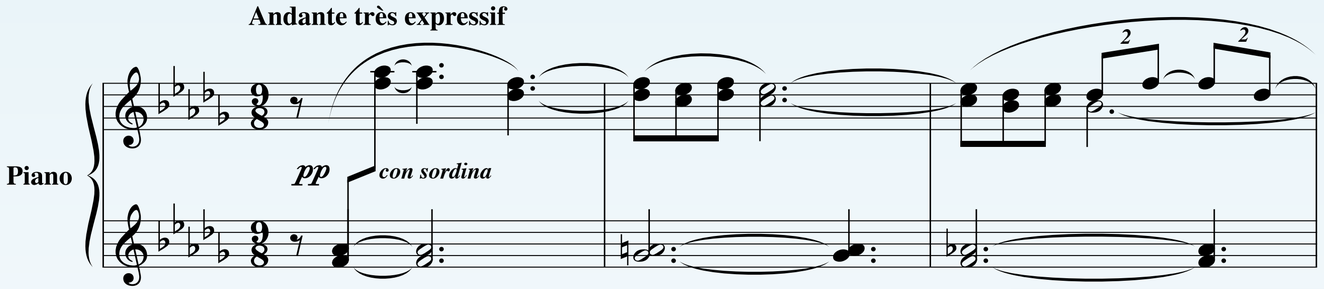
Algebraic
1. e4 e5
2. Nf3 Nf6
3. Bb5 a6
4. Bxc6 dxc6

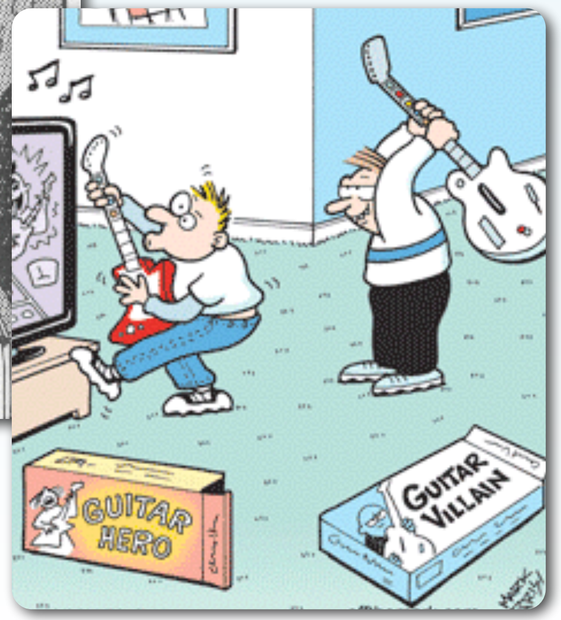
ICCF
1. 5254 5755
2. 7163 2836
3. 6125 1716
4. 2536 4736

Example Domain: Music

Many Notations ...

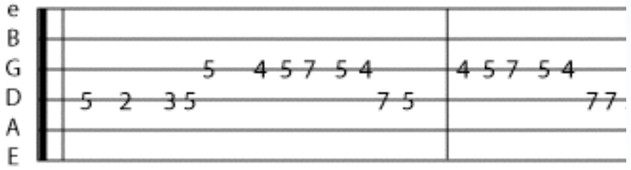
(1862-1918)





Edim9

C#m7+



DSLs vs. Programming Languages

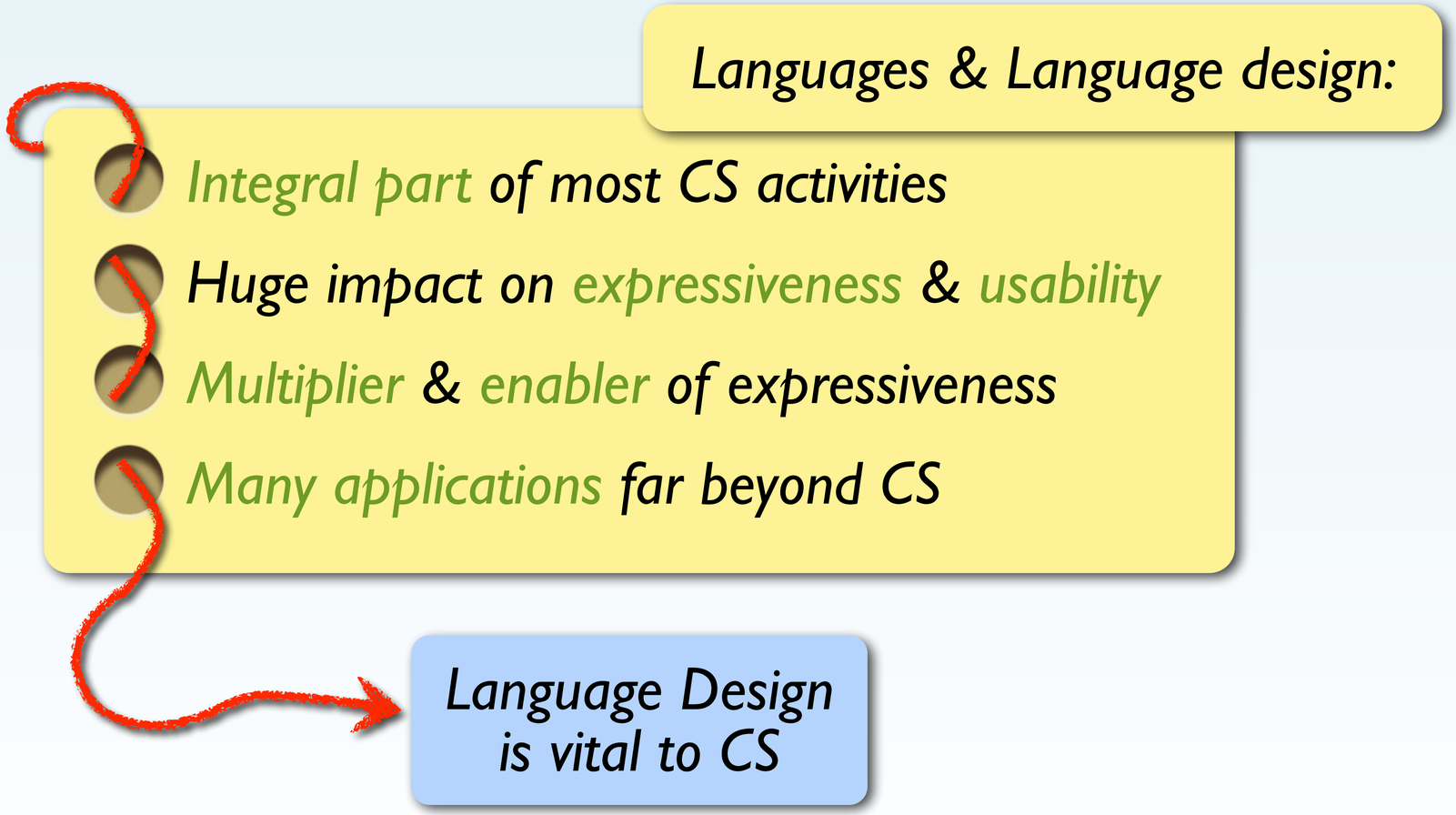
Program: Description of a computation

Programming Language: Set of programs

⇒ *A programming language defines a range of computations*

complete: *Java, C, Haskell, ...*
partial: *Domain-specific languages*
(SQL, Excel, make, Latex, HTML, LabView, ...)

The Reach of Language Design

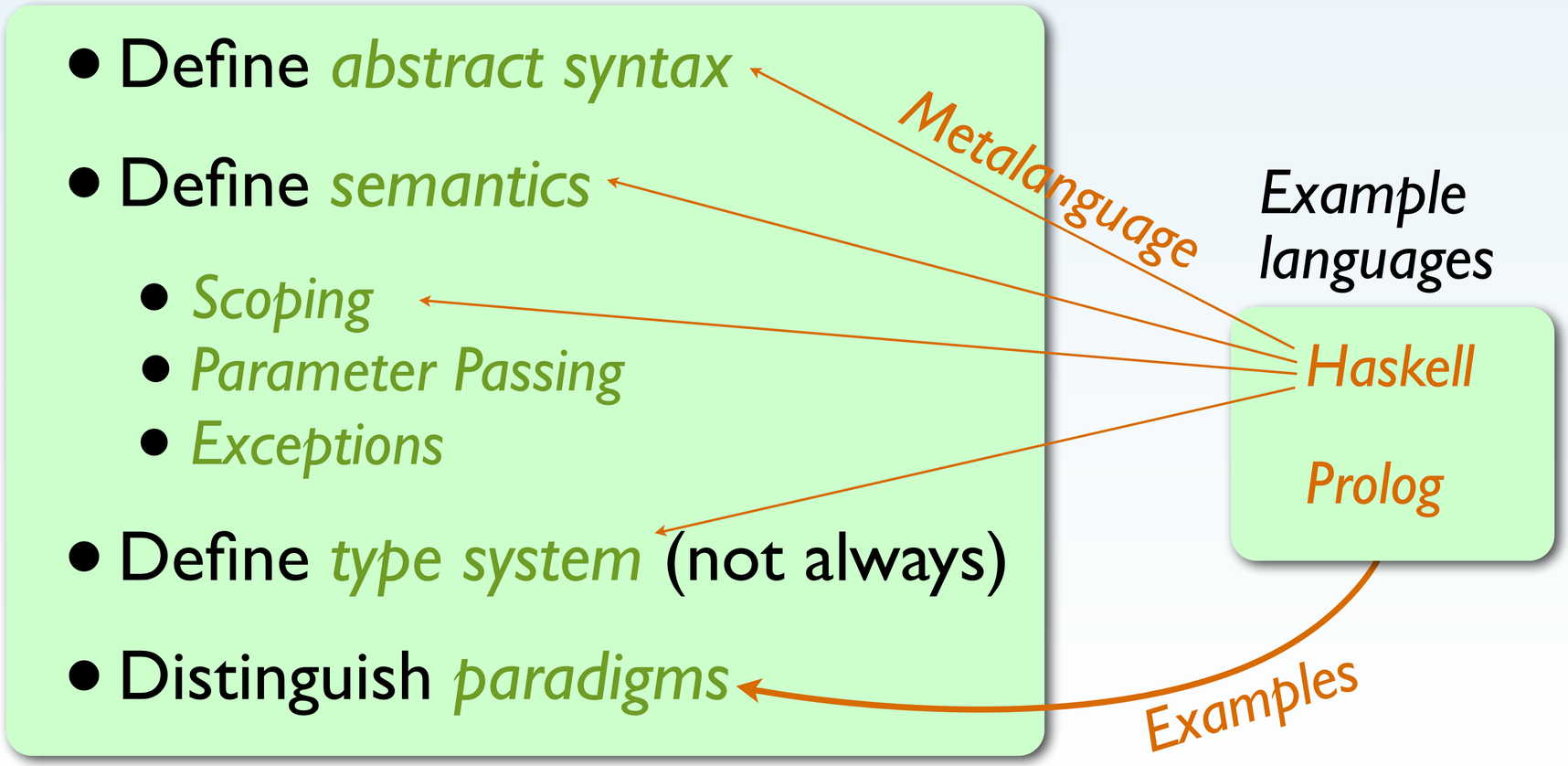


1 Introduction

- About "computer science"
- The role of programming languages
- How to study programming languages?

Approach & Tools

Elements of programming language definition & understanding:



Learning Objectives

	Class Section								
	1 Haskell	2 Abstract Syntax	3 Semantics	4 Types	5 Scope	6 Parameter Passing	7 Exceptions	8 Paradigms	9 Prolog
1 Abstract Syntax		✓	✓	✓					
2 Static & Dynamic Scoping					✓	✓			
3 Static & Dynamic Typing				✓				✓	
4 Parameter Passing						✓		✓	
5 Runtime Stack					✓	✓	✓		
6 Polymorphism				✓			✓	✓	
7 Exception Handling							✓		
8 Paradigms	✓	✓	✓			✓		✓	✓
9 Semantics			✓	✓	✓	✓	✓	✓	

1 Introduction

- About "computer science"
- The role of programming languages
- How to study programming languages?
- 381 Learning objectives and class topics
- Review

Review

- *Define "computation"*
- *What is a program? A programming language?*
- *Explain importance of metalanguages*
- *Why study Haskell?*
- *Name steps involved in defining a new language*
- *Classify learning objectives into 2 different kinds*
- *What is learning objective 7? (just kidding)*

Review

- *Define "computation"*
  Systematic transformation of representations.
- *What is a program? A programming language?*
  P: Description of computation. PL: Tool for describing computations.
- *Explain importance of metalanguages*
  Metalanguages are needed for the description of languages.
- *Why study Haskell?*
  Expressive metalanguage and example for functional paradigm.
- *Name steps involved in defining a new language*
  (Abstract) syntax, (type system), semantics.
- *Classify learning objectives into 2 different kinds*
  Steps of language definition, example languages to illustrate paradigms.

Studying Programming Languages

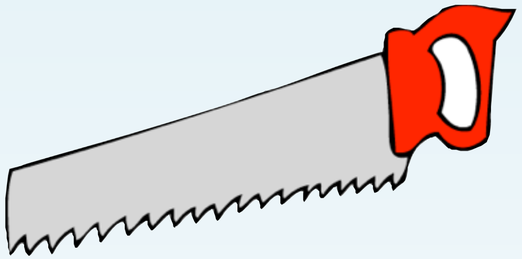
A programming language provides a particular way to describe computations

Programming Language
Tool for describing computations

How to pick a tool (i.e. a language)?

⇒ *Compare features!*

Comparing Tools



concrete & simple

judgment based on physical experience

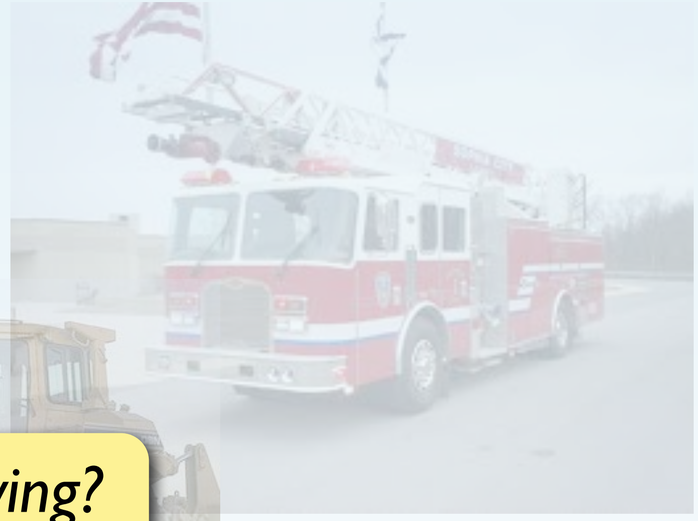
abstract & complex

syntax, static & dynamic semantics, scoping, pragmatics

Programming language theory sharpens your senses

Exploring Vehicles

Exploring Vehicles



Comparing by driving?

Vehicle Features

Description Mechanisms

Vehicle Features

engine
transmission
wheels
doors
safety elements
entertainment system
...

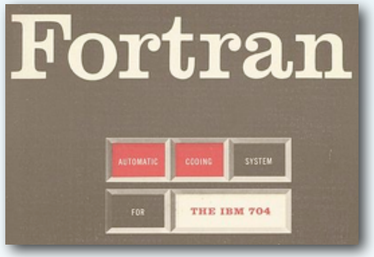
Lots of math

...

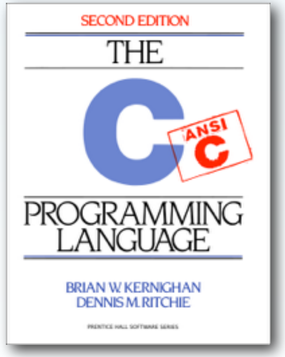
Vehicle Aspects

form, color, style
function (speed, fuel economy, load)
how to operate in different situations
usage profile (sedan, SUV, ...)

Language Diversity

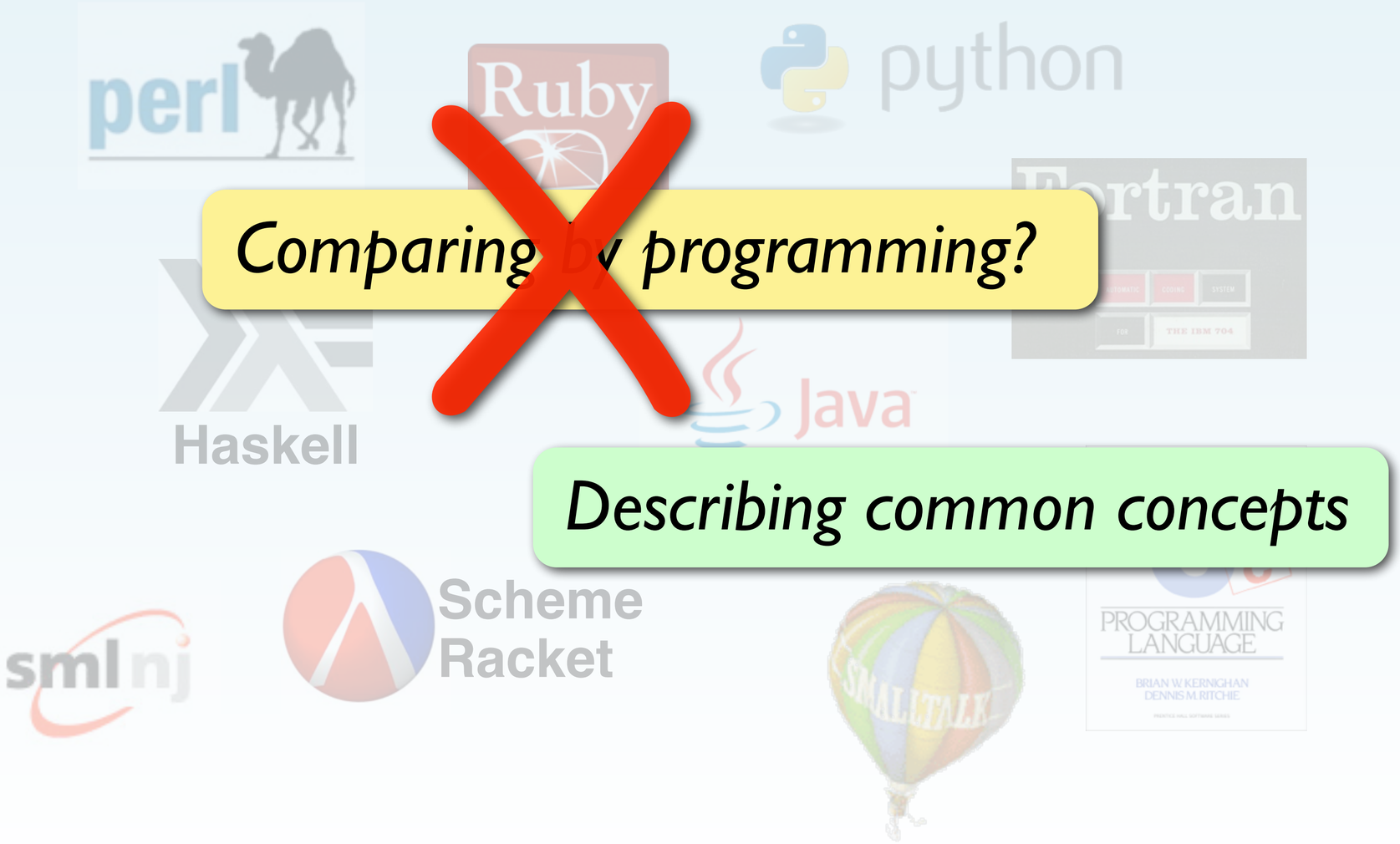


Haskell



Scheme
Racket

Exploring Languages



~~*Comparing by programming?*~~

Describing common concepts

Exploring Languages

Language Features

- values
- operations
- names
- functions
- data types
- state
- typing

Language Aspects

- English
- ✓ Grammars
- Rule systems
- ✓ Haskell,
- Prolog, ...
- Lambda Calculus
- ✓ Runtime stack
- ...
- ✓ syntax (form)
- ✓ semantics (meaning)
- pragmatics (use)
- ✓ paradigm (feature sets)

Metalanguages & Computational Models

Language Processing

- parsing
- ✓ type checking
- compiling
- ✓ interpreting