

# NPRG075

Making programming  
easier and learnable

Tomáš Petříček, 309 (3rd floor)

✉ [petricek@d3s.mff.cuni.cz](mailto:petricek@d3s.mff.cuni.cz)

➔ <https://tomasp.net> | [@tomaspetricek](#)

Lectures: Monday 12:20, S7

➔ <https://d3s.mff.cuni.cz/teaching/nprg075>







# Introduction

Programming for non-programmers

# What & why

## Programming for non-programmers

-  Augmenting human intellect research theme
-  Reducing costs of programming for businesses
-  Computer science & general education
-  Thinking about how to think when programming!


2022 CSforALL Summit Registration Open.

About Member Login Membership Info JROTC-CS SCRIPT Curriculum Directory Upcoming Events

**CSforALL** Projects and Programs Member Directory Donate News and Media

**Join the Movement to Bring Computer Science to ALL Students**

I am a



# Computational thinking

Is that teaching everyone to code?

What to teach and how to best do it?

Designing languages for education?

## LOGO (1967)



Characteristics of the era

Not just a programming language for kids

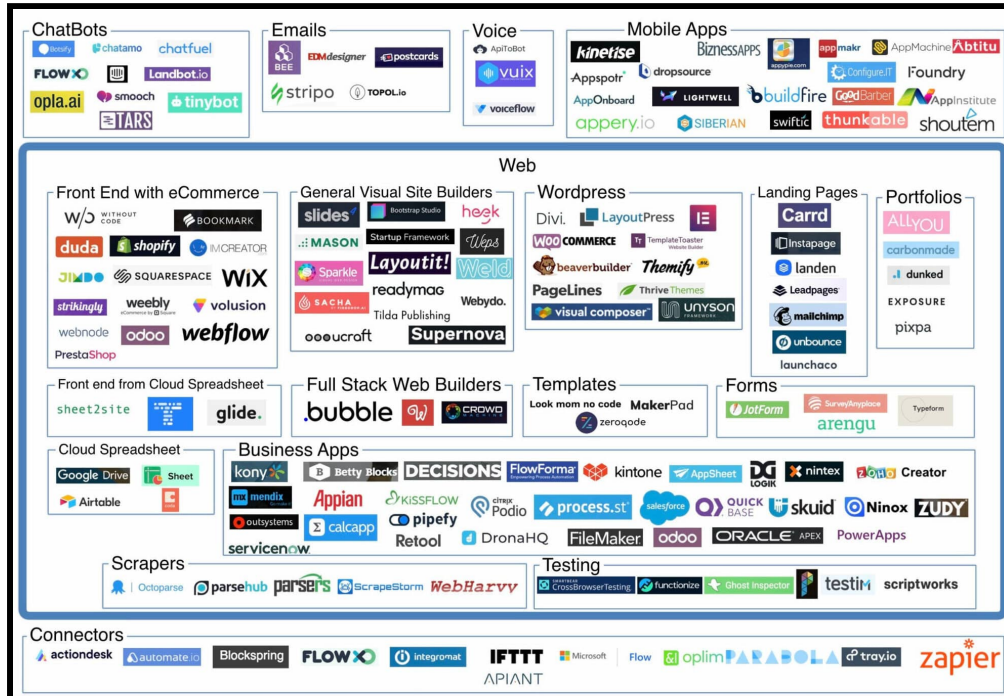
Computer environment: people, things, ideas

Computer culture: a way of thinking about thinking

# No-code and low-code

Platforms for creating applications with minimal code

A new take on end-user programming



UNIQUE SAVINGS  
of the  
**UNIVAC**  
**FLOW-MATIC**  
SYSTEM



**1**

**Virtually Eliminates Your Coding Load**

Your skilled programmers are freed from clerical drudgery to do more creative work. **FLOW-MATIC** shifts emphasis of the programming effort from detailed coding to problem definition and systems analysis. Slashes drastically the time required to program new or altered **UNIVAC** applications.





# FLOW-MATIC

High-level business oriented predecessor of COBOL (1957)

Makes coding so easy your company will not need programmers!

# Methodology

## Programming for non-programmers

-  Metaphors for explaining programming
-  Cognitive models to understand human thinking
-  Finding more manageable kinds of interactions
-  Understanding & assisting with common errors



# End-user programming

Making programming super easy

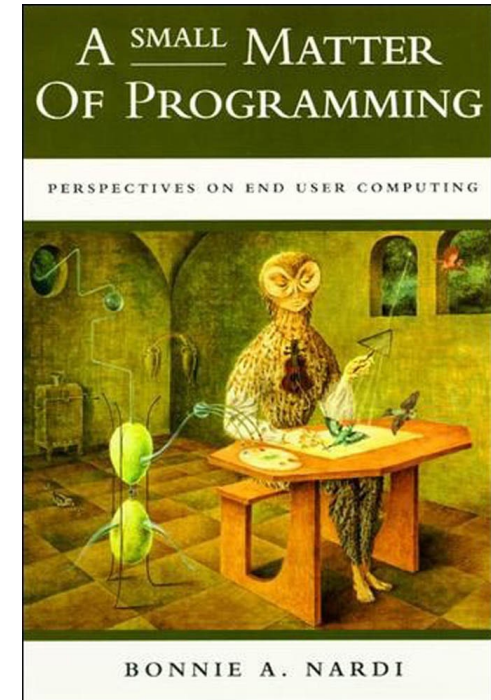
# A small matter of programming

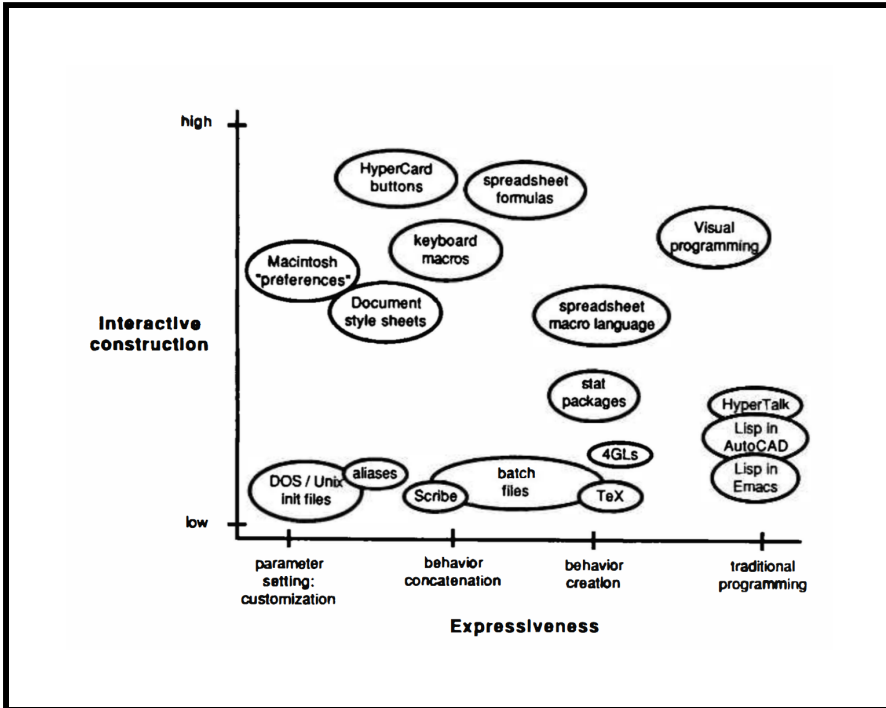
## End-user programming (1993)

- Spreadsheets, CAD systems, statistical packages
- Task specific systems

## An elusive dream?

- Can anyone become a programmer?
- Beyond task-specific?
- Programmable end-user systems?





# End-user programming

- ① Very high-level  
Domain-specific languages
- ② Spreadsheets  
CAD & statistical systems
- ③ User interaction  
New kinds of specifying

# High-level languages

## FLOW-MATIC (1960s)

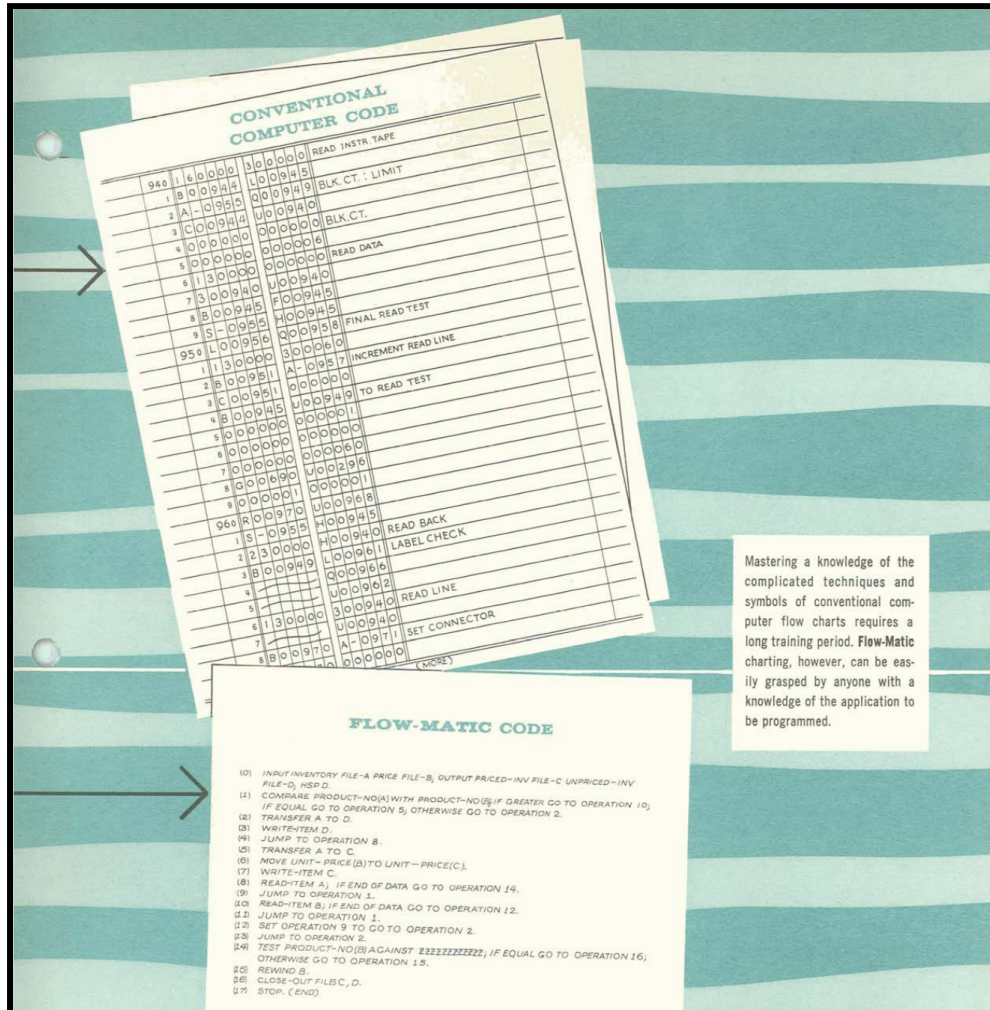
English; easily taught to clerical workers

## DSLs (2000s)

Small languages for specific problems

## Low-code (2020s)

GUI-based entire app development



# Case study: Darklang

Domain-specific  
abstractions for  
server-less backends

- HTTP handler
- Worker
- Database
- CRON job

```
HTTP GET /coding-challenge/outlier/timeseries
DB::query `Timeseries \value -> value.country = request.queryParams.country`
```

```
HTTP GET /coding-challenge/outlier/import
emit/1 "request" "importTimeseriesWorker"
```

```
WORKER importTimeseriesWorker
let _ = importTimeseries "australia"
let _ = importTimeseries "usa"
importTimeseries "france"
```





DB Timeseries.v0

All entries are identified by a unique string 'key'.

date	Date
location	String
value	Int
country	String

# Notations

## Limits of high-level notations

-  Requires a "tidy" problem domain
-  There is no universal language
-  Adaptable notations tend to be complex
-  Cannot (should not?) accept human vagueness

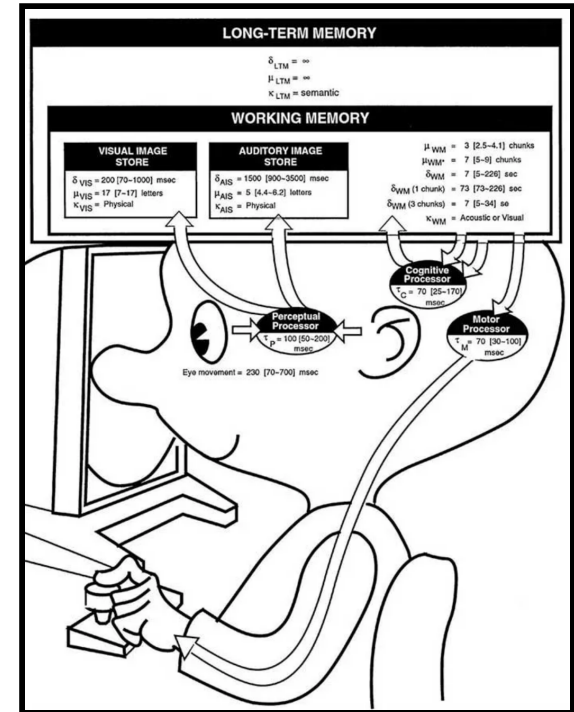
# What makes programming hard?

## Cognitive obstacles

- Loss of direct manipulation (and the frame problem)
- Use of (specialized) notation
- Abstraction for complexity

## Attention investment model

- Cognitive obstacles have cost
- Programming as an investment
- When is the gain worth it?



# Eliminating cognitive obstacles

## Spreadsheet-based interfaces

Avoid abstraction and give immediate feedback

## Programming by example

No need for notation and abstraction

## Direct manipulation

Manipulate concrete entities & post-hoc abstraction



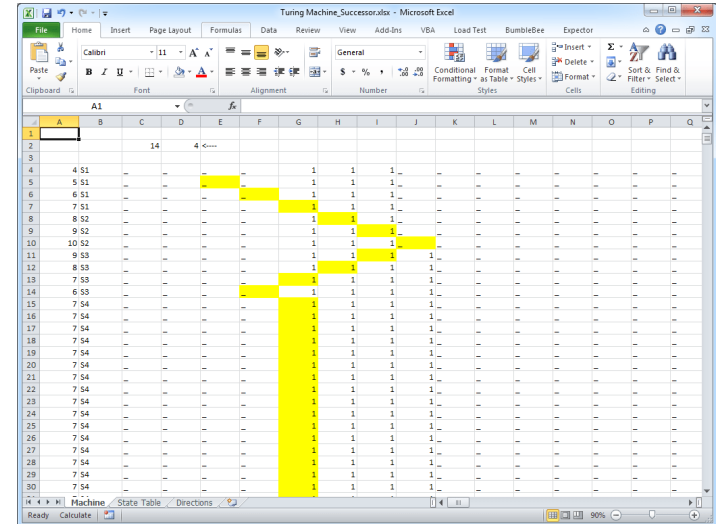
# Spreadsheets as programming

Are they really programming?

- Domain-specific, but powerful
- Turing-complete (in a way)
- Lambdas, macros, extensions

Spreadsheets & programming

- IDEs can learn about liveness
- Spreadsheets can learn about software engineering
- TechDims: Abstraction construction, feedback loops



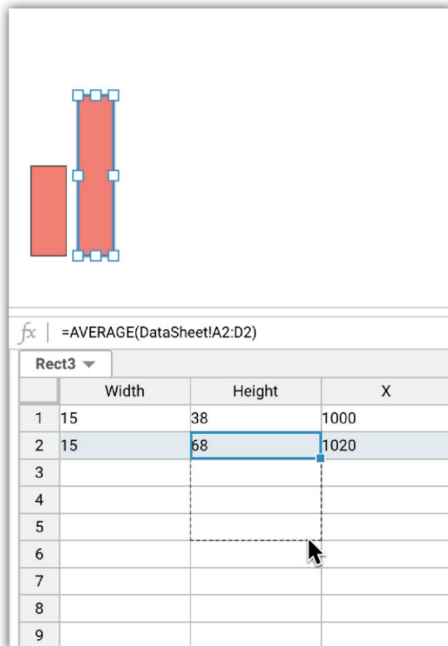
# General-purpose spreadsheets?

(Marasoiu, 2019)

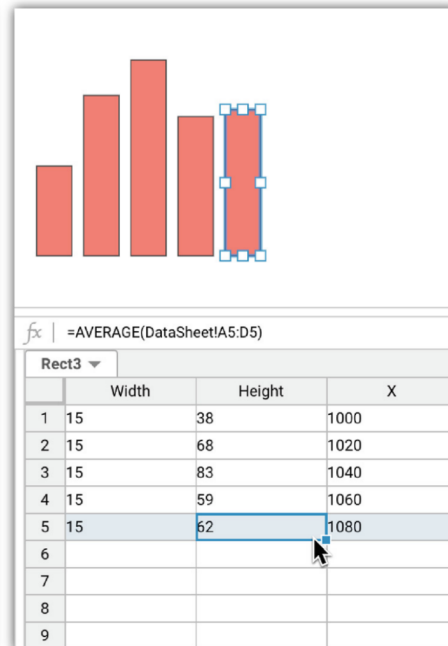
Spreadsheet-based data visualization

Spreadsheet interface for constructing custom charts

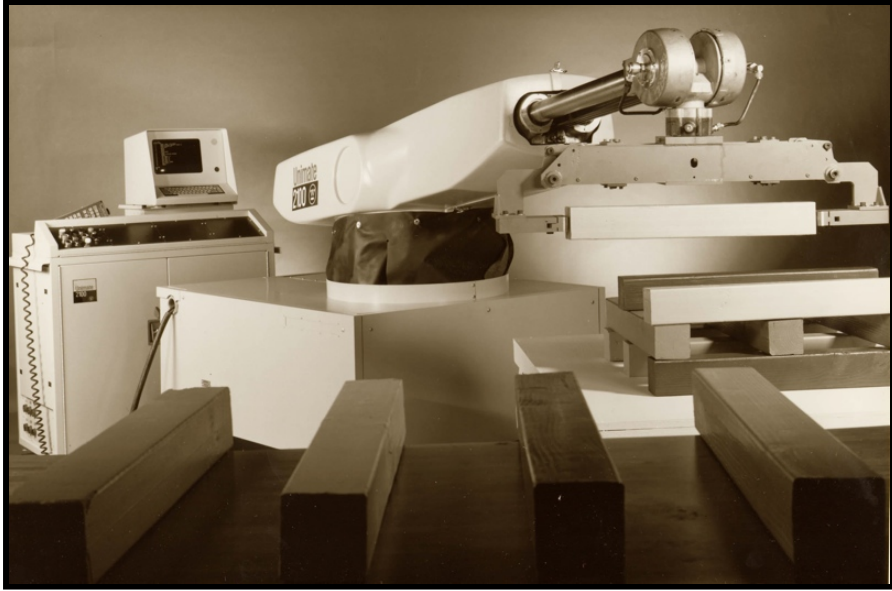
What else could we express this way?



*A.*



*B.*



## Direct manipulation

Complete task manually,  
have computer repeat it

Industrial robots, graphics  
editing, task automation,  
geometry, formatting

How to allow for small  
variation in behaviour?

# Wrangler

(Kandel et al, 2011)

Data wrangling by  
direct manipulation

User cleans with data

System builds a script

Attempts to generalize  
concrete interactions

The screenshot shows the Apache Wrangler interface. On the left is a 'Transform Script' panel with a list of operations: 'Split data repeatedly on newline into rows', 'Split split repeatedly on ''', 'Promote row 0 to header', 'Delete empty rows', 'Extract from Year after 'in'', 'Set extract's name to State', and 'Fill State by copying values from above'. Below the script are tabs for 'Text', 'Columns', 'Rows', 'Table', and 'Clear'. The main area displays a table with columns 'Year', 'State', and '# Property'. The table contains data for reported crime rates in Alabama, Alaska, and Arizona for the years 2004, 2005, 2006, 2007, and 2008. The first row (index 0) is highlighted in red and contains the text 'Reported crime in Alabama' in the Year column and 'Alabama' in the State column. The second row (index 1) contains '2004', 'Alabama', and '4029.3'. The third row (index 2) contains '2005', 'Alabama', and '3900'. The fourth row (index 3) contains '2006', 'Alabama', and '3937'. The fifth row (index 4) contains '2007', 'Alabama', and '3974.9'. The sixth row (index 5) contains '2008', 'Alabama', and '4081.9'. The seventh row (index 6) is highlighted in red and contains the text 'Reported crime in Alaska' in the Year column and 'Alaska' in the State column. The eighth row (index 7) contains '2004', 'Alaska', and '3370.9'. The ninth row (index 8) contains '2005', 'Alaska', and '3615'. The tenth row (index 9) contains '2006', 'Alaska', and '3582'. The eleventh row (index 10) contains '2007', 'Alaska', and '3373.9'. The twelfth row (index 11) contains '2008', 'Alaska', and '2928.3'. The thirteenth row (index 12) is highlighted in red and contains the text 'Reported crime in Arizona' in the Year column and 'Arizona' in the State column. The fourteenth row (index 13) contains '2004', 'Arizona', and '5073.3'. The fifteenth row (index 14) contains '2005', 'Arizona', and '4827'. The sixteenth row (index 15) contains '2006', 'Arizona', and '4741.6'. The seventeenth row (index 16) contains '2007', 'Arizona', and '4502.6'. The eighteenth row (index 17) contains '2008', 'Arizona', and '4087.3'. The nineteenth row (index 18) is highlighted in red and contains the text 'Reported crime in Arkansas' in the Year column and 'Arkansas' in the State column. The twentieth row (index 19) contains '2004', 'Arkansas', and '4033.1'. The twenty-first row (index 20) contains '2005', 'Arkansas', and '4068.1'.

	Year	State	# Property
0	Reported crime in Alabama	Alabama	
1	2004	Alabama	4029.3
2	2005	Alabama	3900
3	2006	Alabama	3937
4	2007	Alabama	3974.9
5	2008	Alabama	4081.9
6	Reported crime in Alaska	Alaska	
7	2004	Alaska	3370.9
8	2005	Alaska	3615
9	2006	Alaska	3582
10	2007	Alaska	3373.9
11	2008	Alaska	2928.3
12	Reported crime in Arizona	Arizona	
13	2004	Arizona	5073.3
14	2005	Arizona	4827
15	2006	Arizona	4741.6
16	2007	Arizona	4502.6
17	2008	Arizona	4087.3
18	Reported crime in Arkansas	Arkansas	
19	2004	Arkansas	4033.1
20	2005	Arkansas	4068.1

# Programming by example

## FlashFill and FlashExtract

- Write (or select) examples
- System infers patterns
- Refine examples to clarify

## Implementation

- Synthesize programs to match
- Using carefully chosen small language
- And a suitable search algorithm

```
DLZ - Summary Report
"Sample ID.: ""5007-01""
"Sample Date/Time.: ""Wednesday, May 30, 2006 00:43:51""
Intensities
"/S: ""Analyte"" ""Mass"" ""Conc. Mean"" ""Unit"" ""Conc. SD"" ""RSD"" ""Mean""
"/, ""Be"" 9,0.070073, ""ug/L"" 0.009,12.542,121.334"
"/> ""Sc"" 45, ""ug/L"" 404615.043"
"/, ""Ti"" 48,10.653153, ""ug/L"" 0.847,7.949,181379.200"
"/, ""Se"" 82,1.009204, ""ug/L"" 0.026,2.613,457.487"
"/, ""Sr"" 88,20.163079, ""ug/L"" 2.005,9.943,718014.023"
"/> ""Rh"" 103, ""ug/L"" 438976.176"

DLZ - Summary Report
"Sample ID.: ""5007-02""
"Sample Date/Time.: ""Wednesday, May 30, 2006 01:02:38""
Intensities
"/S: ""Analyte"" ""Mass"" ""Conc. Mean"" ""Unit"" ""Conc. SD"" ""RSD"" ""Mean""
"/, ""Mn"" 55,71.705740, ""ug/L"" 0.350,0.489,2428667.736"
"/, ""Co"" 59,0.131132, ""ug/L"" 0.004,3.315,3606.816"
"/, ""Ba"" 138,129.339264, ""ug/L"" 3.088,2.387,4648771.382"
"/, ""Hf"" 178, ""ug/L"" 338359.496"
"/, ""Tl"" 205,2.876992, ""ug/L"" 0.730,25.380,129217.588"
"/, ""Pb"" 208,3.671043, ""ug/L"" 0.026,0.702,228830.402"
```

# Education

Teaching programming & thinking

# MIT Artificial Intelligence Lab

## Minsky & Papert

"Seymour Papert and Marvin Minsky thought about thinking, about children's thinking and about machine's thinking."



## LOGO project & language

- Computers as "native speakers" of mathematics
- Teach creative and logical thinking
- Giving children tools to learn (Montessori)

# LOGO as a language

## Language features

- Interactive and LISP-inspired
- Lists, recursion, functional
- More of an idea than a language

## LOGO for education

- Learning through microworlds
- Give kids the most powerful language created
- Powerful ideas: anthropomorphization, metalanguage





```
TO NOUN
  OUTPUT PICK [BIRDS DOGS ..]
END
TO VERB
  OUTPUT PICK [HATE BITE LOVE]
END
TO ADJECTIVE
  OUTPUT PICK [RED PECULIAR ..]
END

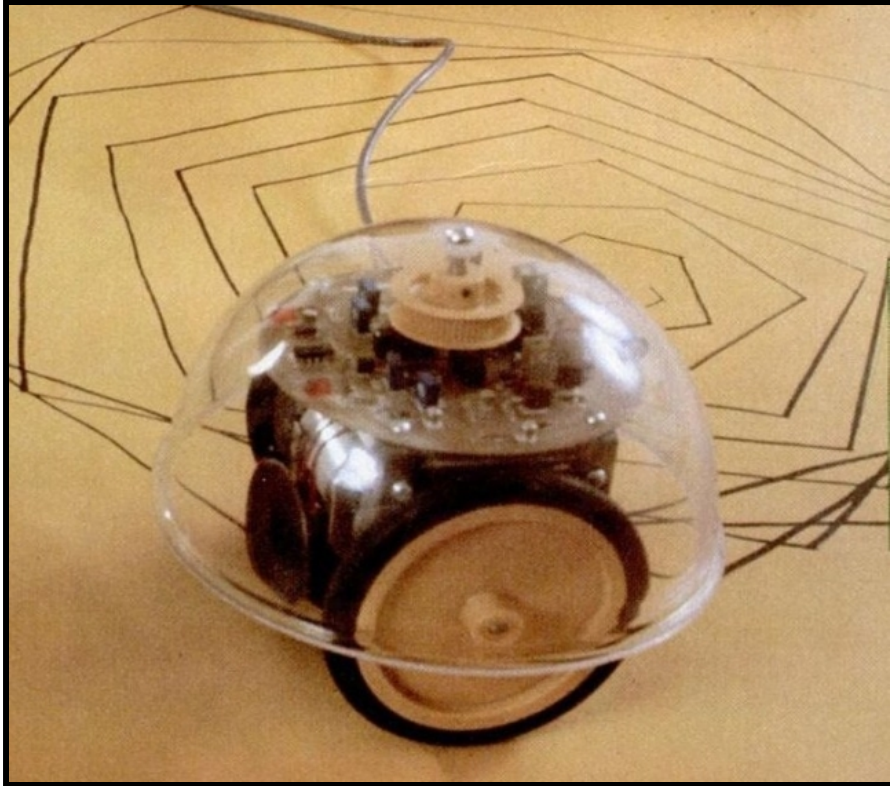
PRINT (SENTENCE ADJECTIVE
      NOUN VERB ADJECTIVE NOUN)
```

## Microworlds

A small domain-specific language for exploring ideas

Turtle graphics is best known example

First LOGO example was for word plays



## Turtle microworld

On-screen and floor robots





Great for teaching

Debug by pretending to be the turtle & follow program

Does not blame students ("the turtle has a bug")

# Computer science education

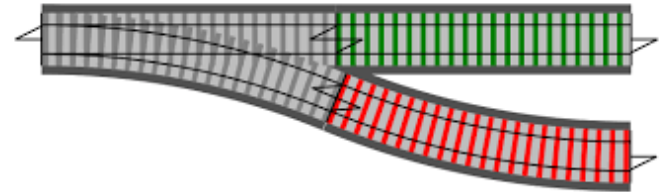
## Teaching programming thinking today

-  From 1960s idealism to 2020s pragmatism
-  Focus on what we can convincingly study
-  Improving teaching practices & methods
-  Developing better conceptual frameworks

# Notional machines

## Models for thinking

- Model of a computer operation
- Helps understand computation
- A "useful lie" for teaching



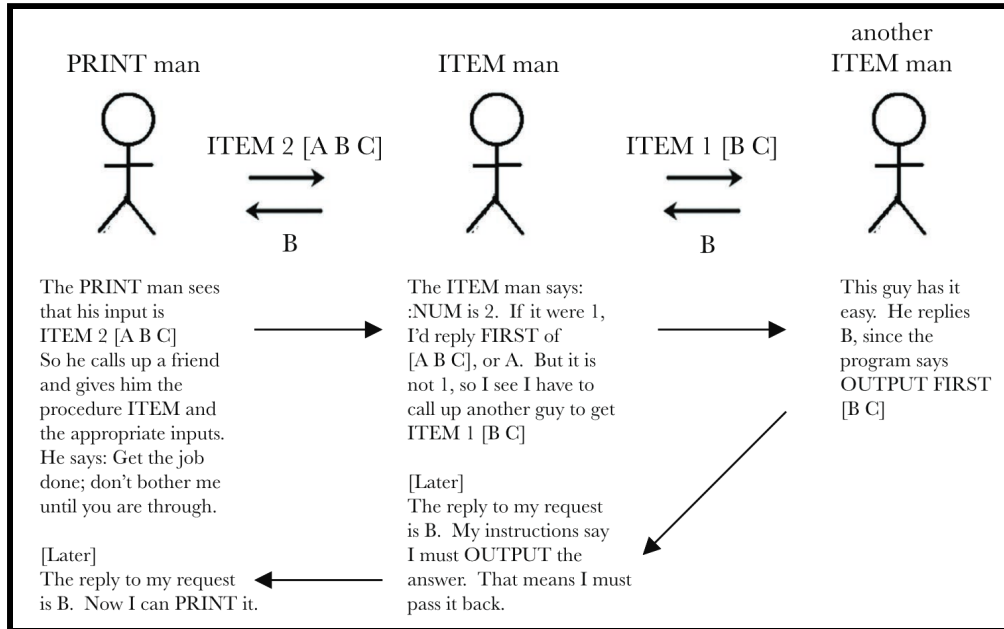
## Example notional machines

- Objects and message passing of Smalltalk
- LOGO "little people" metaphor
- Computation as railway track

# Little people metaphor

A powerful idea for understanding how programs work

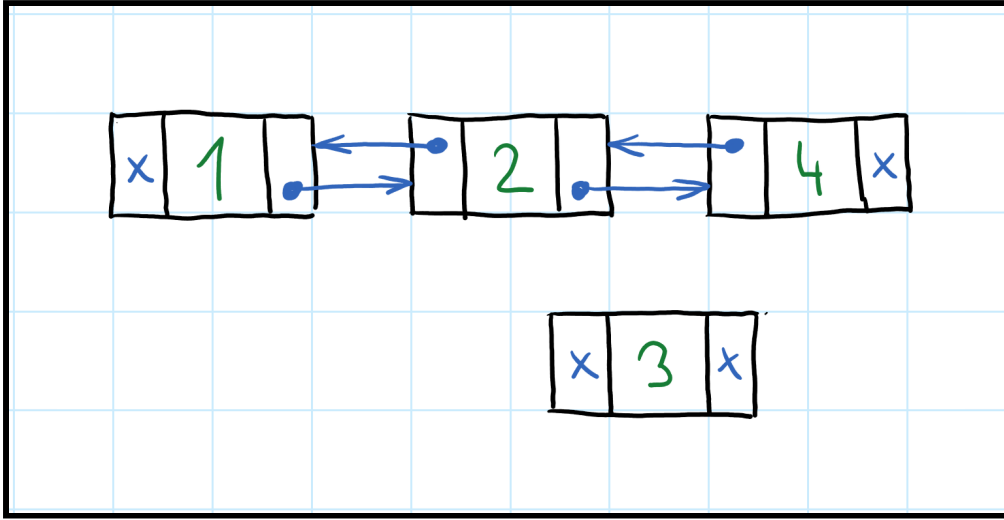
Function instantiation as a "little men" doing (one step of) work



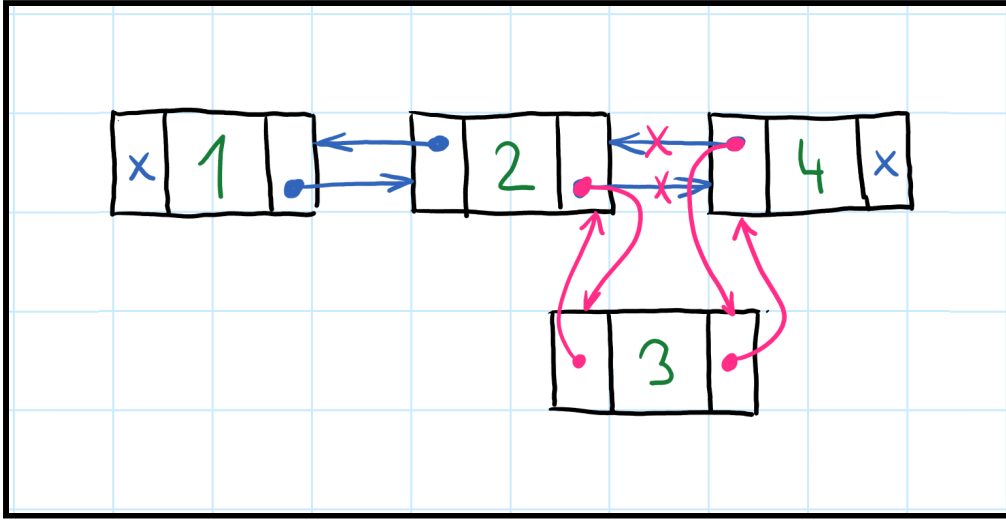
## Linked lists (1/2)

Boxes with pointers as connecting arrows

Let's insert 3 in the list between 2 and 4...



## Linked lists (2/2)







Boxes with pointers as connecting arrows

Let's insert 3 in the list between 2 and 4...

Useful but does not explain everything that pointers can do!

# Computing education

Basic disagreements about the problem

-  Computational thinking & algorithms for all?
-  Creativity as with LOGO and Sonic Pi?
-  History and philosophical problems?
-  How to best teach present-day technology?



# Metaphors

Thinking about programming

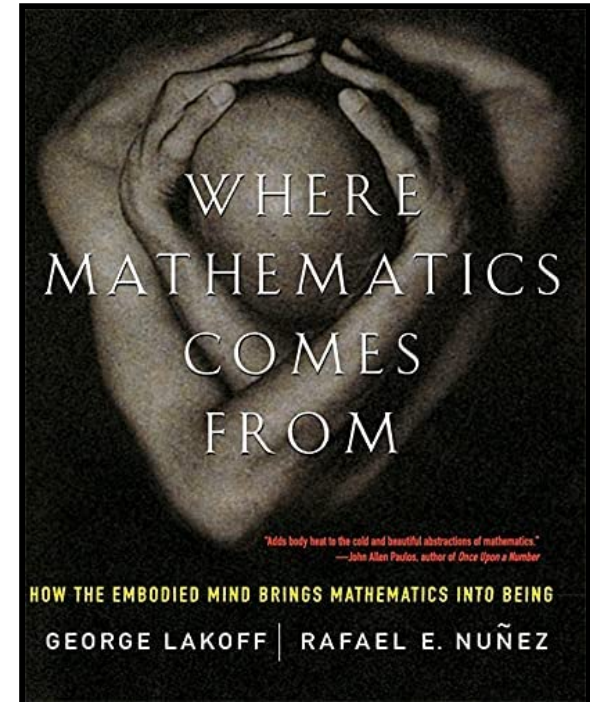
# Metaphors for programming

## Essence of human thought?

- Time as resource, Up as positive, ...
- Apparent through our language
- Basic for constructing mathematics?
- Each has fits and misfits

## Metaphors for programming

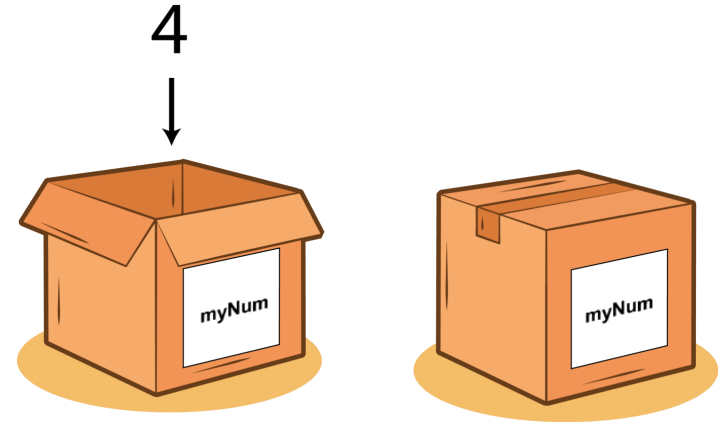
- Notional machines (LISP, Smalltalk)
- Thinking about variables, monads



# Two metaphors for variables

## Variable as a box

- You store value in a box
- Variable "contains" a value
- What is stored in a **name**?



## Variable as a label

- Label you place on a value
- Variable "is" a value
- What is a **name**?

# Misconceptions

Does the metaphor for variables matter?

« What is the meaning of multiple assignment?

☞ Box can contain multiple values!

⊕ Label will be for computation or addition

🏆 Box metaphor wins, but beware of misfits

```
class Monad m where
  (>>=)  ::
    m a -> (a -> m b) -> m b
  return ::
    a -> m a
```

## Metaphors for monads

Interface capturing a class of computations

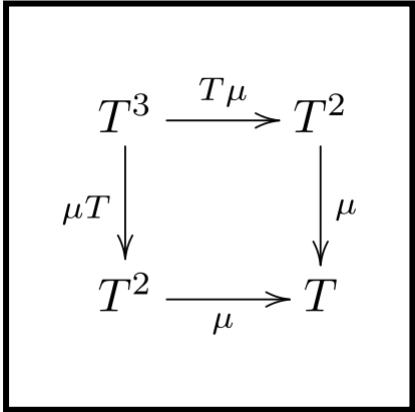
Used for effectful computations in Haskell

How programmers think about them?

# Three metaphors for monads

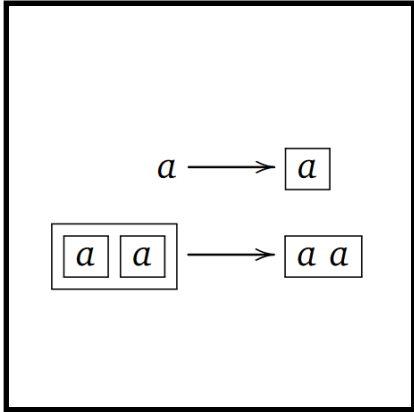
## Symbolic

Meaningless  
symbolical entity  
satisfying laws



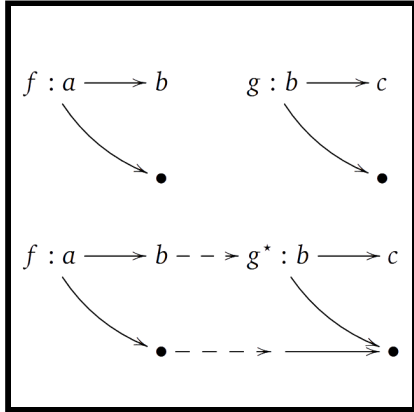
## Box

Container that can  
be transformed  
and un-nested







## Track

Computation that  
can proceed in  
multiple ways



# Misconceptions

## Common errors in thinking

-  Loops terminate when condition turns false
-  Sequential statements do not wait
-  Variable name has effect on its behaviour
-  Missing else branch stops program

# Conclusions

Easier and learnable



# Thank you!

Please do keep in touch!

- Do a final project (and get credit as a bonus)
- Sign-up for a follow-up seminar
- Get in touch about MSc or PhD projects

Tomáš Petříček, 309 (3rd floor)

✉ [petricek@d3s.mff.cuni.cz](mailto:petricek@d3s.mff.cuni.cz)

➔ <https://tomasp.net> | [@tomaspetricek](https://twitter.com/tomaspetricek)

➔ <https://d3s.mff.cuni.cz/teaching/nprg075>

# References (1/3)

## End-user programming

- UNIVAC FLOW-MATIC (1957). [Introducing a new language for automatic programming](#). Sperry Rand Corporation
- Bonnie A. Nardi (1993). [A Small Matter of Programming](#). MIT
- Blackwell, A. F. (2002). [First Steps in Programming: A Rationale for Attention Investment Models](#). VL/HCC
- Blackwell, A.F., Burnett, M. (2002). [Applying Attention Investment to End-User Programming](#). VL/HCC

## Spreadsheets

- Marasoiu, M. et al. (2019). [Cuscus: An End User Programming Tool for Data Visualisation](#). IS-EUD

# References (2/3)

## Programming by demonstration

- Smith, D. C. (1977). [Pygmalion: A Computer program to Model and Stimulate Creative Thought](#). ISR
- Kandel, S., et al. (2011). [Wrangler: Interactive Visual Specification of Data Transformation Scripts](#). CHI
- Cypher A (ed.) (1993). [Watch What I Do: Programming by Demonstration](#). MIT

## Programming by example

- Gulwani, S. et al. (2016). [Programming by Examples](#). DSSE
- Vu Le, Gulwani S. (2014). [FlashExtract: A Framework for Data Extraction by Examples](#). PLDI

# References (3/3)

## Programming education

- Solomon, C. et al. (2020). [History of LOGO](#). HOPL
- Papert S. (1980). [Mindstorms: Children, Computers and Powerful Ideas](#). Basic Books
- Fincher, S. A. & Robins A. V. (eds.) (2019). [The Cambridge Handbook of Computing Education Research](#). Cambridge

## Metaphors & misconceptions

- Lakoff, G. & Nunez, R. (2001). [Where Mathematics Come From](#)
- Petricek, T. (2018). [What we talk about when we talk about monads](#)
- Hermans, F. et al. (2018). [Thinking out of the box: comparing metaphors for variables in programming education](#). WiPSCE
- Swidan, A. et al. (2018). [Programming Misconceptions for School Students](#). ICER

