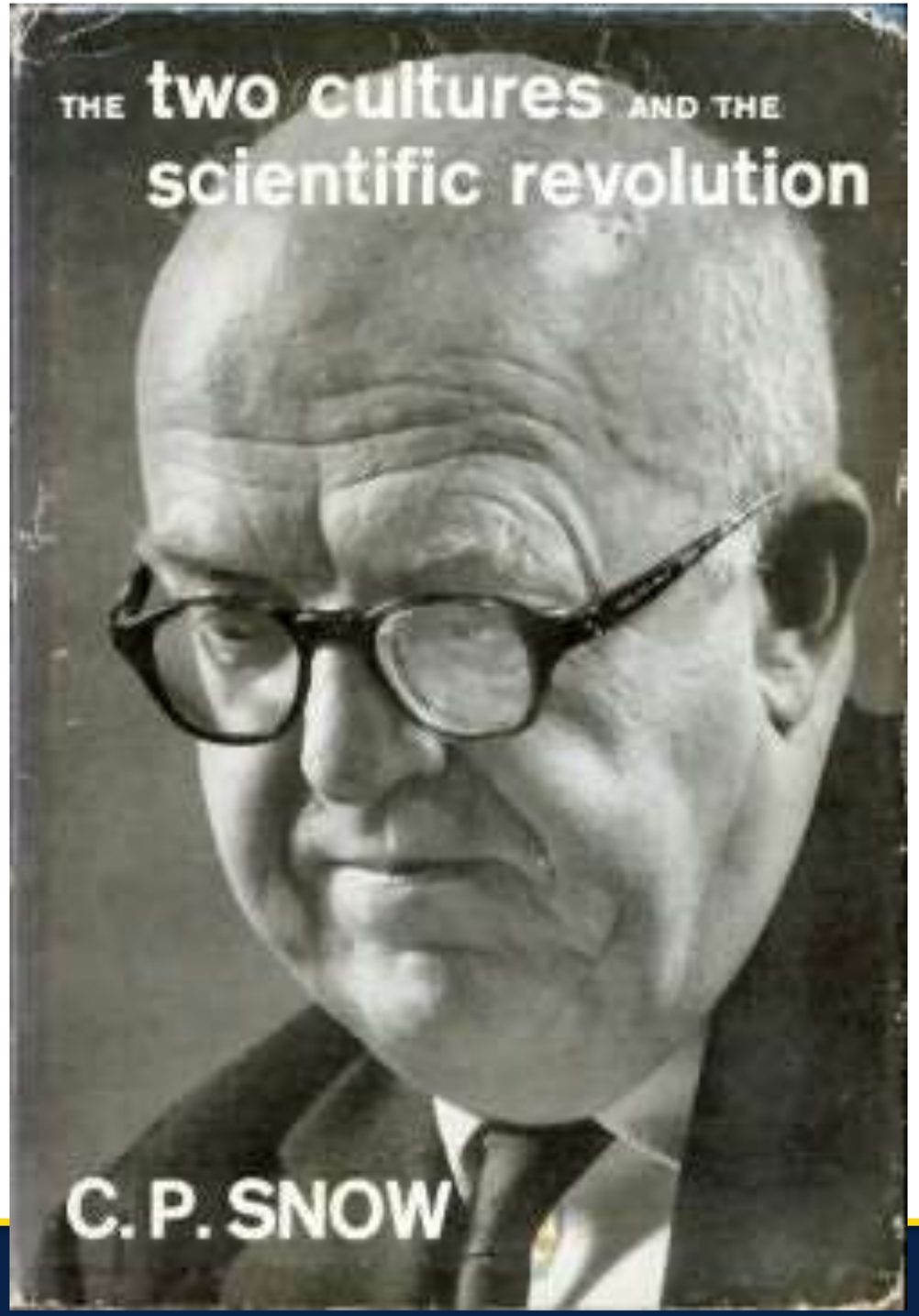
COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF MICHIGAN

Computing Education as a Foundation for 21st Century Literacy

Mark Guzdial (@guzdial) #SIGCSE2019


THE two cultures AND THE
scientific revolution
C. P. SNOW

Special Interest Group on Computer Science Education

Our Mission: To provide a global forum for educators to discuss research and practice related to the learning and teaching of computing…at all education levels

Story

- Computer science was invented to teach *everyone* about everything.
- Computational thinking and the power of the imitation game.
- How computing education will change on the way.
  - What we teach
  - How we'll teach

Definitions

- **Computer science**: The study of computers and all the phenomena associated with them. (Perlis, Newell, and Simon, 1967)

- **Computing**: CS + CE + SE + IS + IT +…
  Computer science facing outward (Denning).

- **Programming**: Reading and writing a notation of computation, a specification of a computer's process.

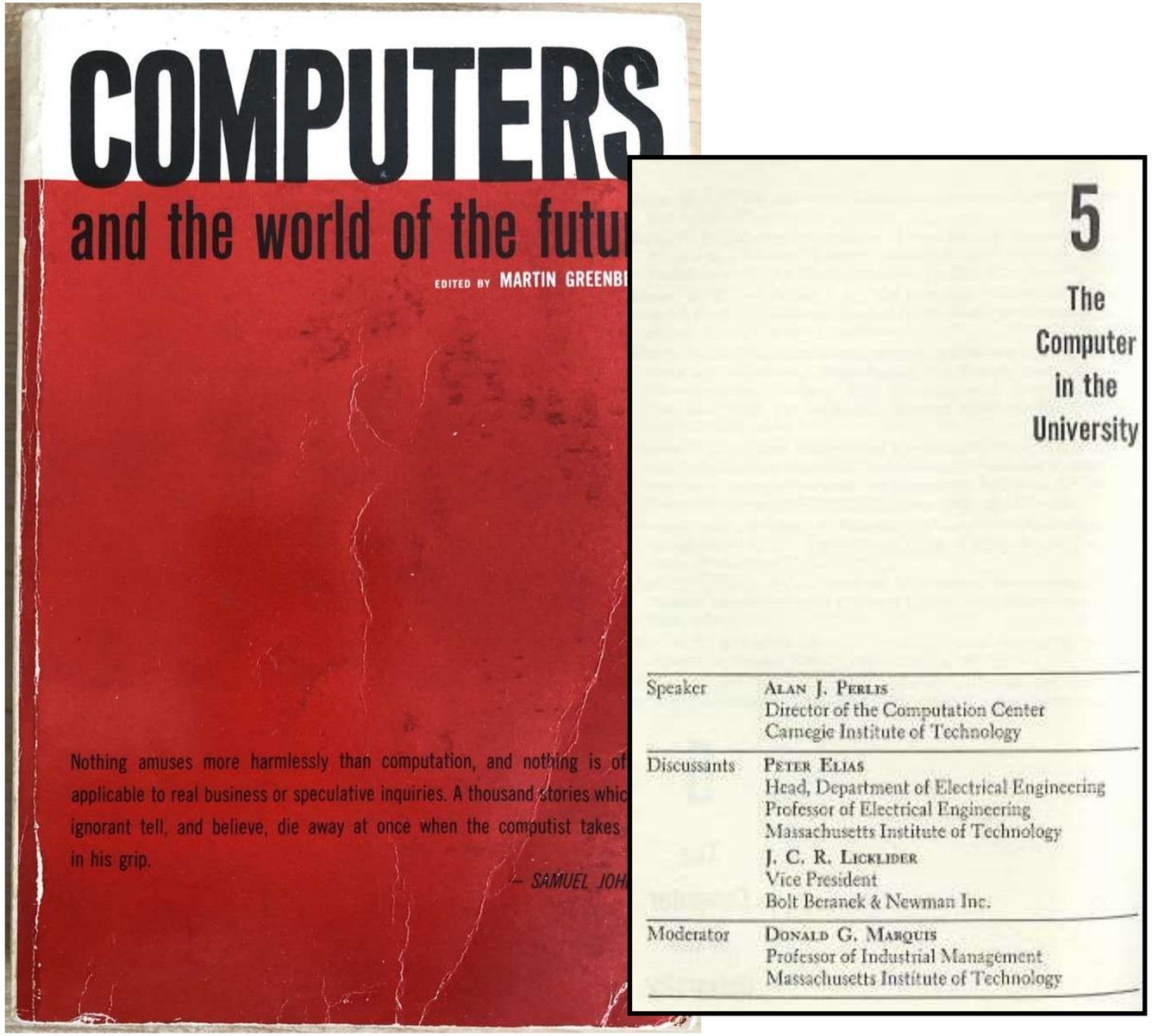
George Forsythe, Inventor of “CS” (1961)

The three most valuable tools in a STEM education (1968):

- Language
- Mathematics
- Computer Science

Computer Science was invented to be a tool for learning everything else

Photo Stanford News

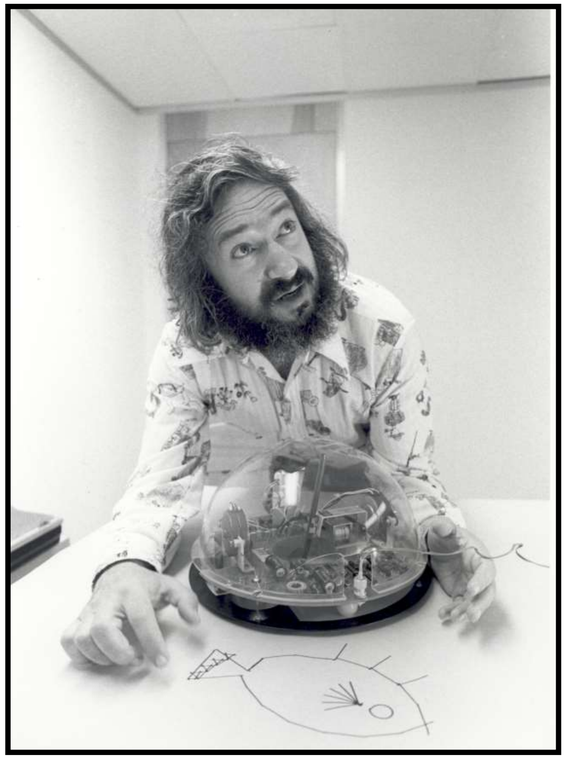


1961

Alan Perlis

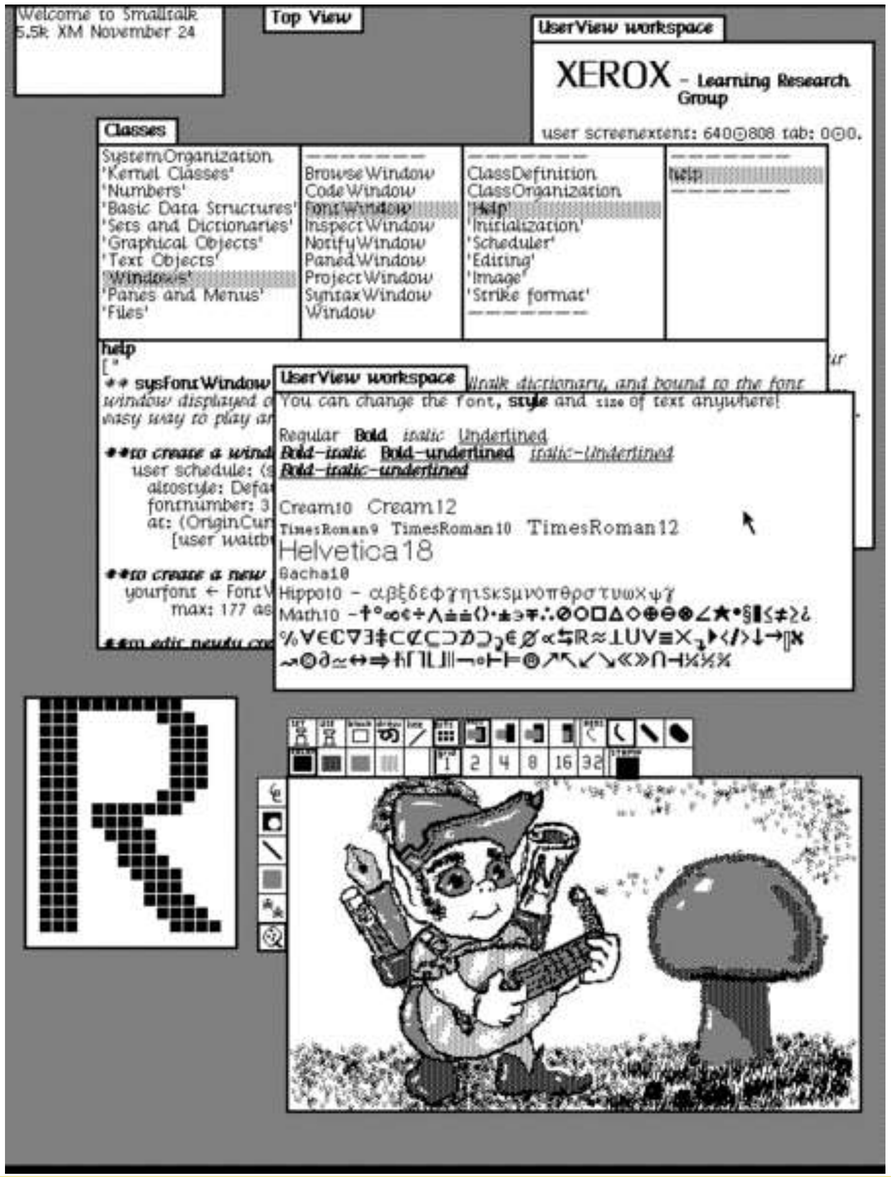
Photo: CMU

Seymour Papert



Thanks to Gary Stager

Kay & Goldberg, “Personal Dynamic Media” (1977)



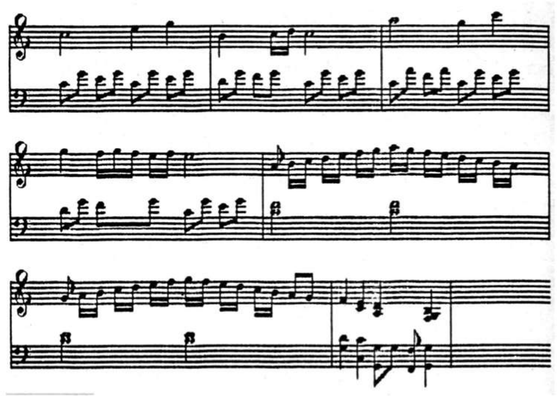


Figure 26.26. Data for this score was captured by playing on a musical keyboard. A program then converts the data to standard musical notation.

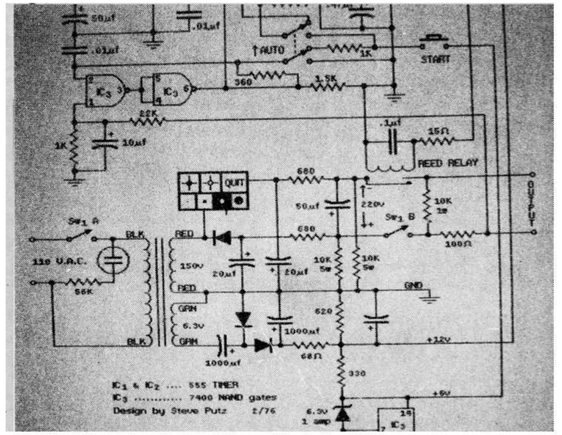
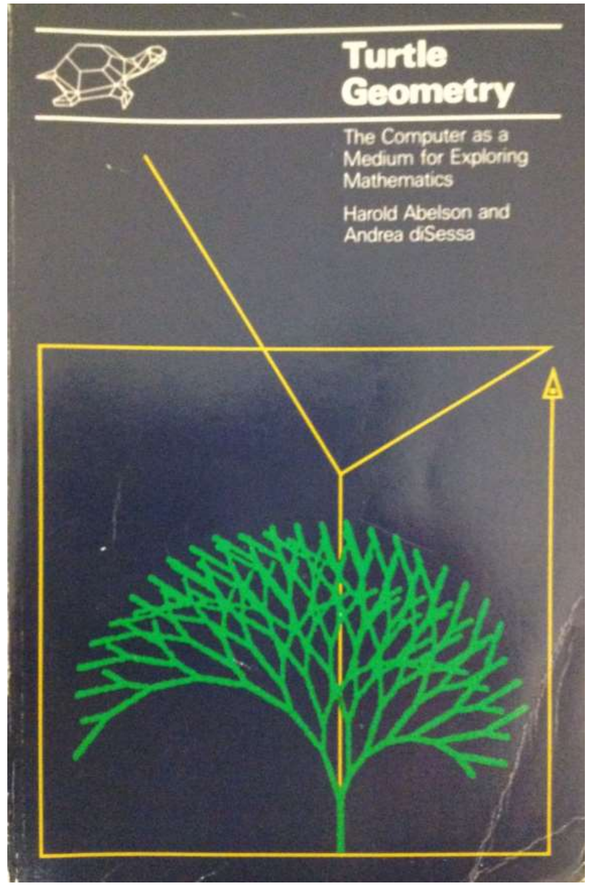
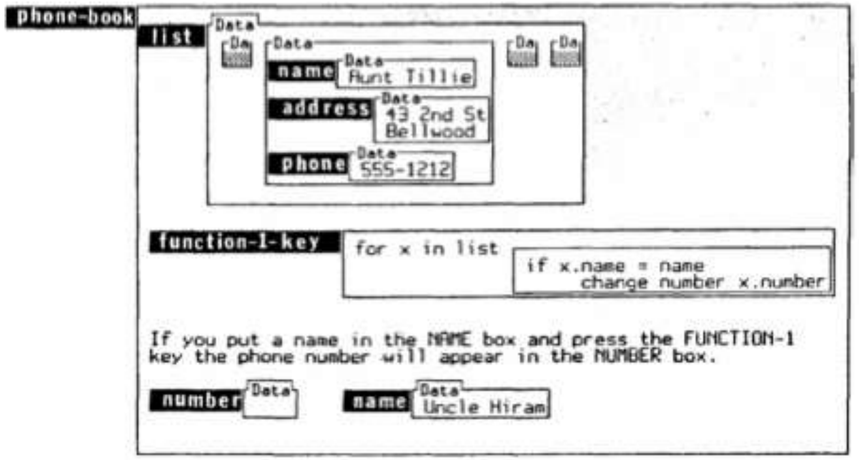
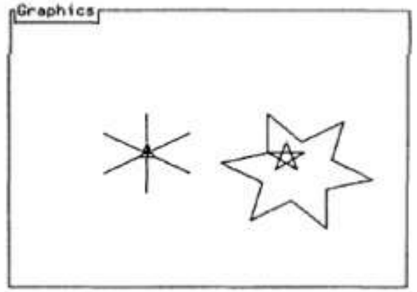


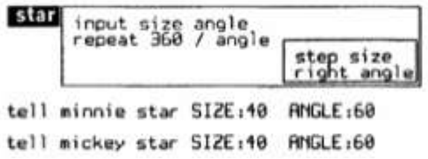
Figure 26.27. An electronic circuit layout system programmed by a 15-year-old student. Circuit components selected from a menu are moved into position with the mouse, connected to other components with lines, and labeled with text. The rectangle in the center of the layout is an iconic menu from which line widths and connectors (solid and open) may be chosen.

Andrea diSessa, Boxer, and Computational Literacy

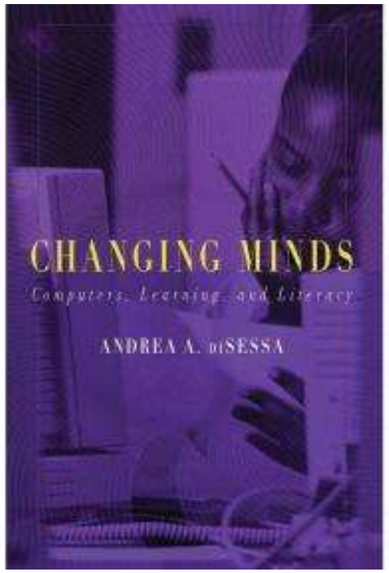




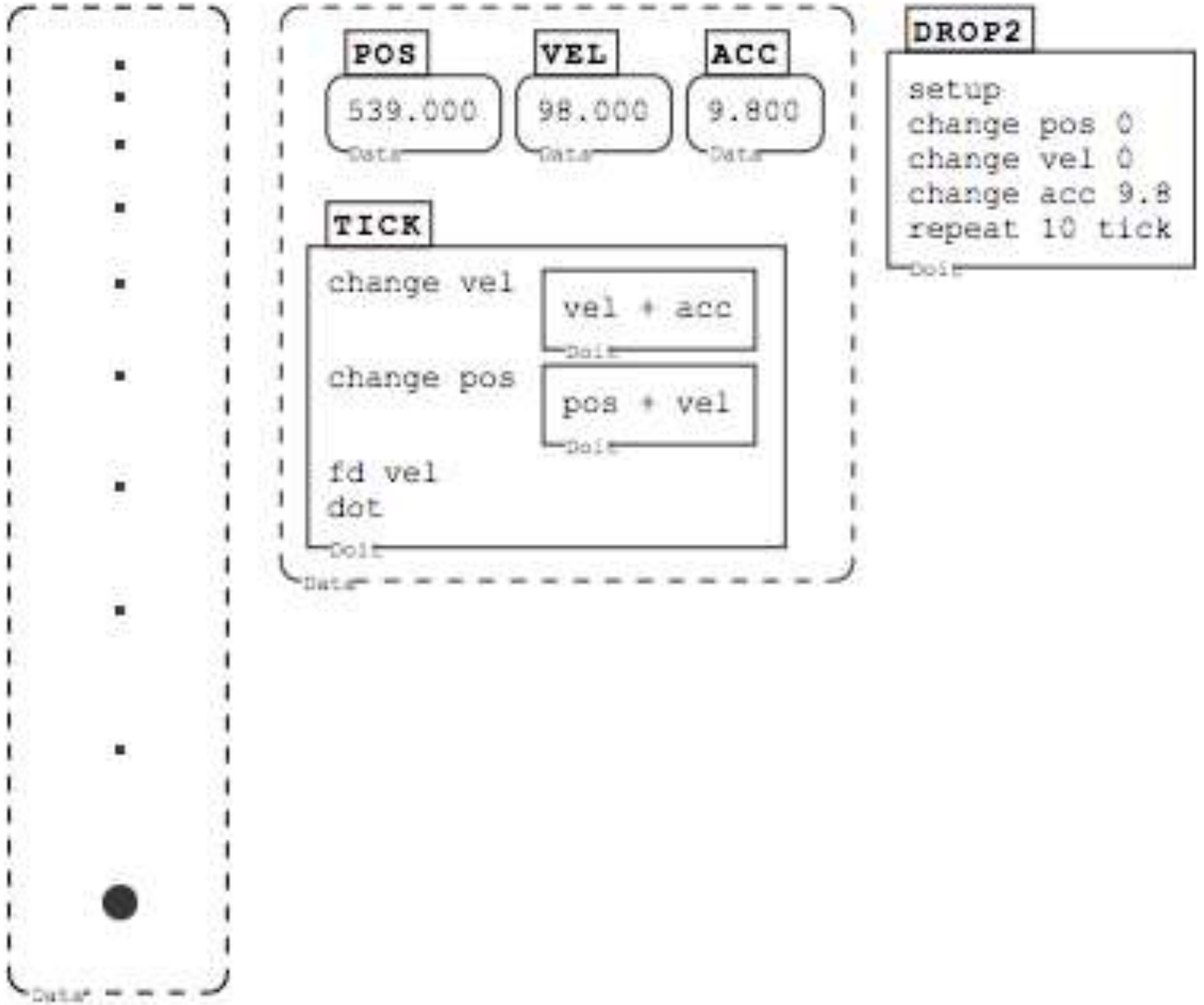





```
tell minnie star SIZE:40  ANGLE:60
tell mickey star SIZE:40  ANGLE:60
```



Code is Different



Bruce Sherin

Code is Causal

Researcher: Can you tell me how long it took the rock to get to the ground?

Student B: It would be about one second

R: Okay, where did you get that from?

B: If the acceleration is 30 feet per second per second, then per second it will be going 30 feet per second, then it will just take a little longer for it to get to the ground.

R: Why?

B: Because you have to divide the, to get the average velocity, which is how fast it's going, and how you can measure how far it's gone, you have to... let's see... it will be going, it will be going 15 meters per second. Maybe two seconds, I guess.

R: Why?

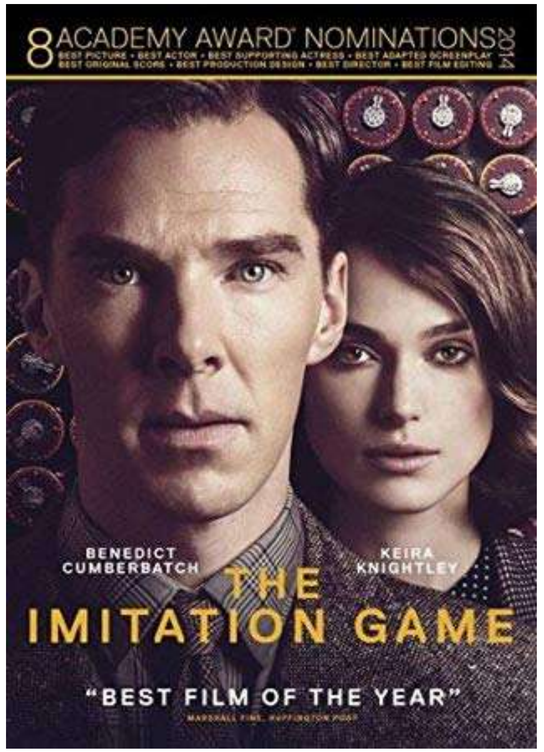
B: Because... 1.5 seconds. Because, by the time it's accelerated the second second, it will be going about 45 feet per second, so it'll have to be between the first and second second that it hits the ground.

Computation as a literacy

- Computing is:
  - A causal specification of process that can be automated.
  - The master simulator.

- Can **be** and **effect** the way that students learn everything else.

- The goal is enjoyable fluency.

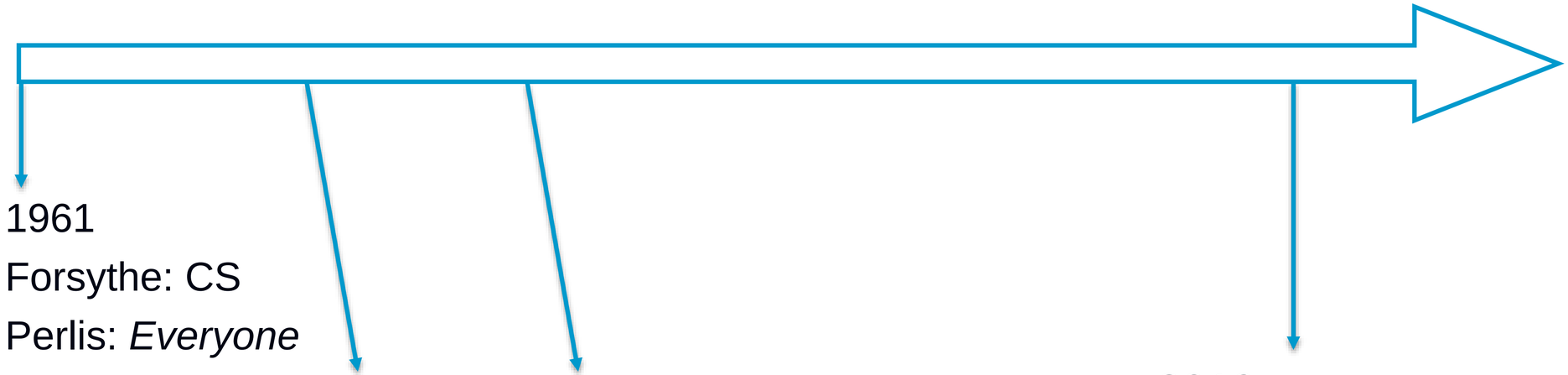


It's what *Computer Science* was created for.

Special Interest Group on Computer Science Education

Our Mission: To provide a global forum for educators to discuss research and practice related to the learning and teaching of computing…**at all education levels**

Our timeline



In Scotland, CS teacher numbers are declining

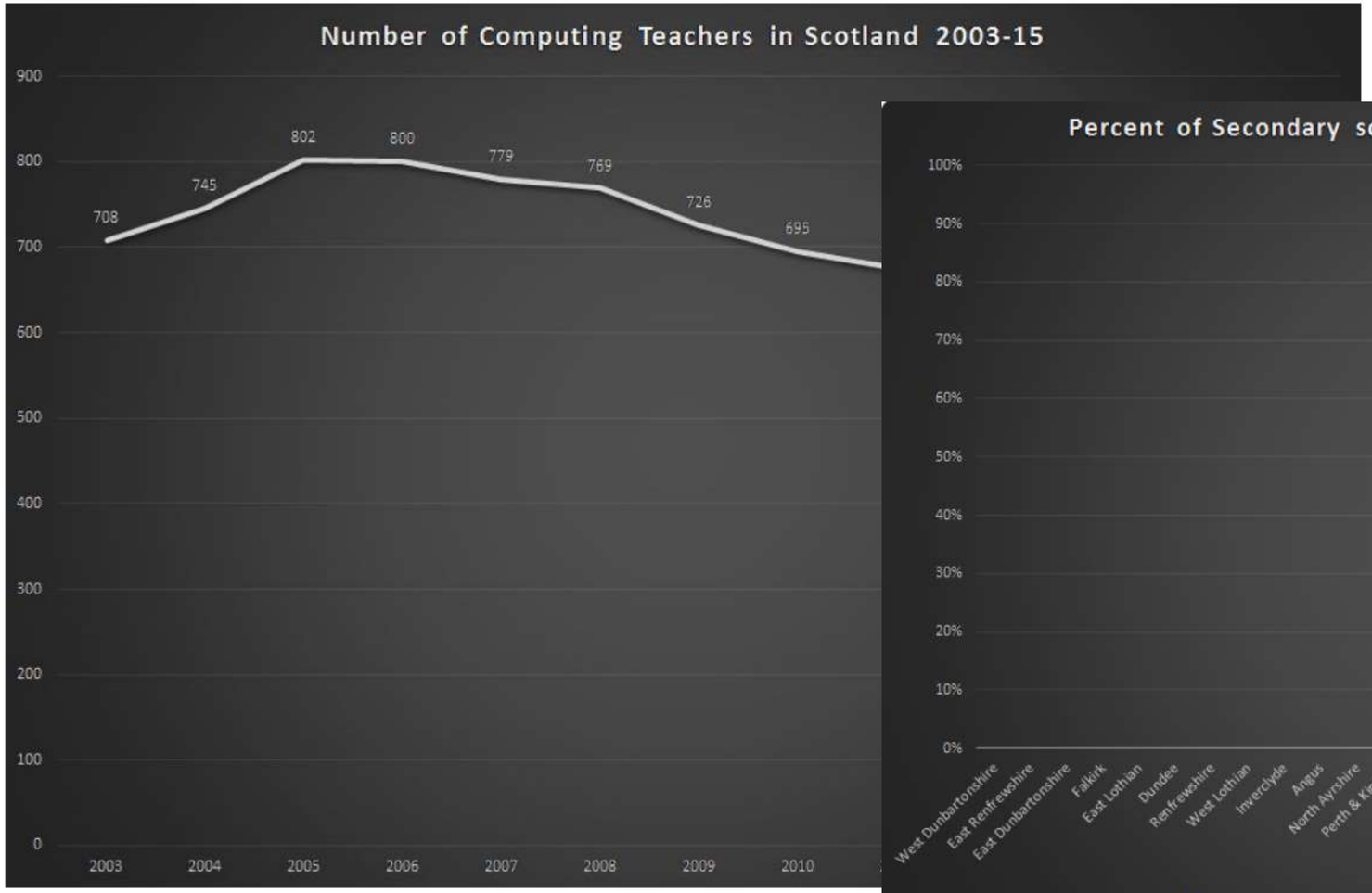


Figure 1: Number of CS Teachers 2003

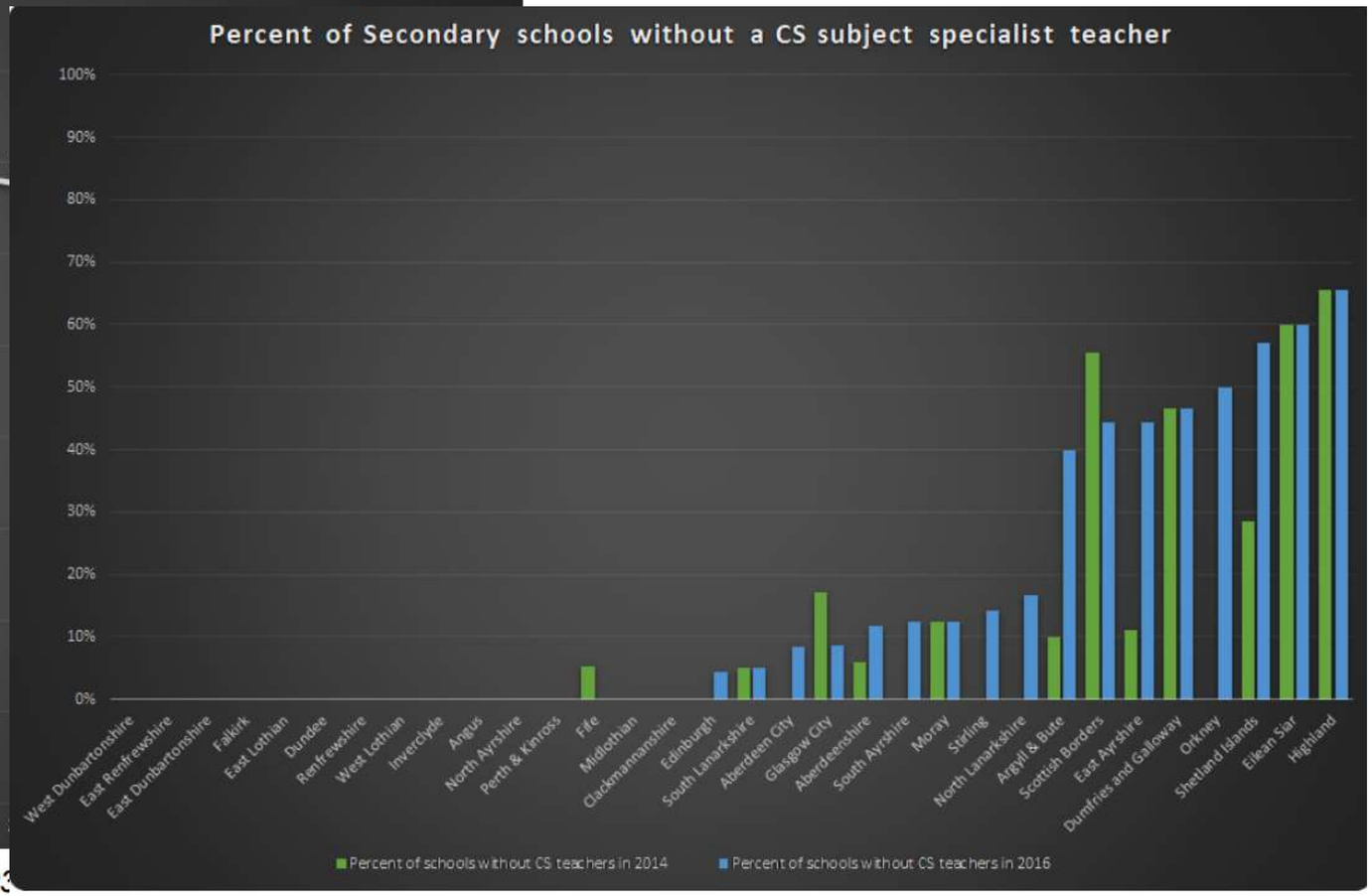
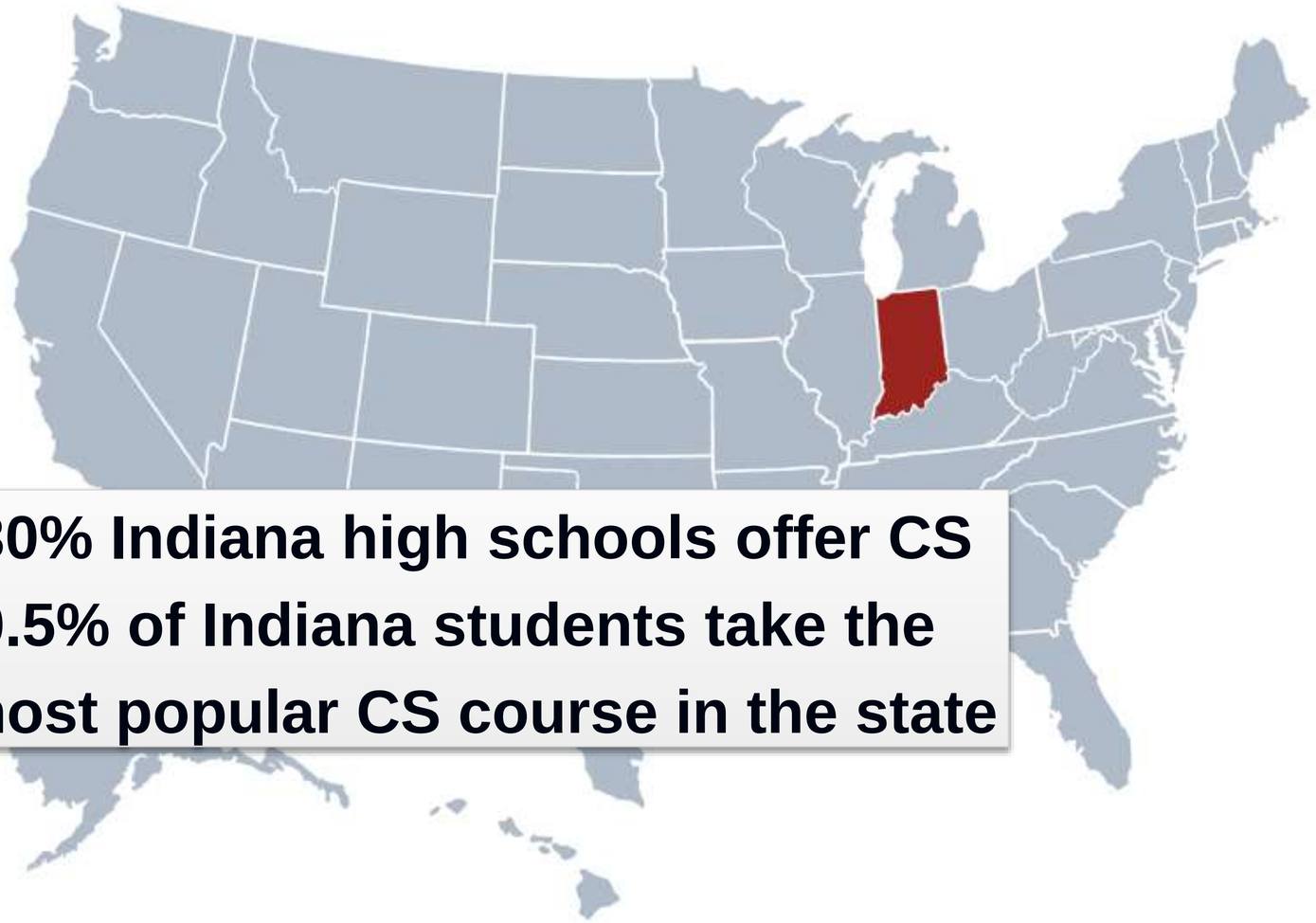


Figure 6: Percent of Secondary Schools without CS teachers in 2014 and 2016

Computer science in high schools is growing very slowly

- In the UK (from Roehampton Report):
  - 53% of schools offer CS GCSE,
    12% of students take it.
    - < 20% female
  - 36% offer A Level CS, under 3% take it.
    - < 10% female

- In US states, few students taking CS.
  By far, mostly male, esp. AP CS (A and CSP).



< 30% Indiana high schools offer CS

< 0.5% of Indiana students take the most popular CS course in the state

Thanks to Anne Leftwich and Miranda Parker

47% of Georgia schools offer CS in 2016.

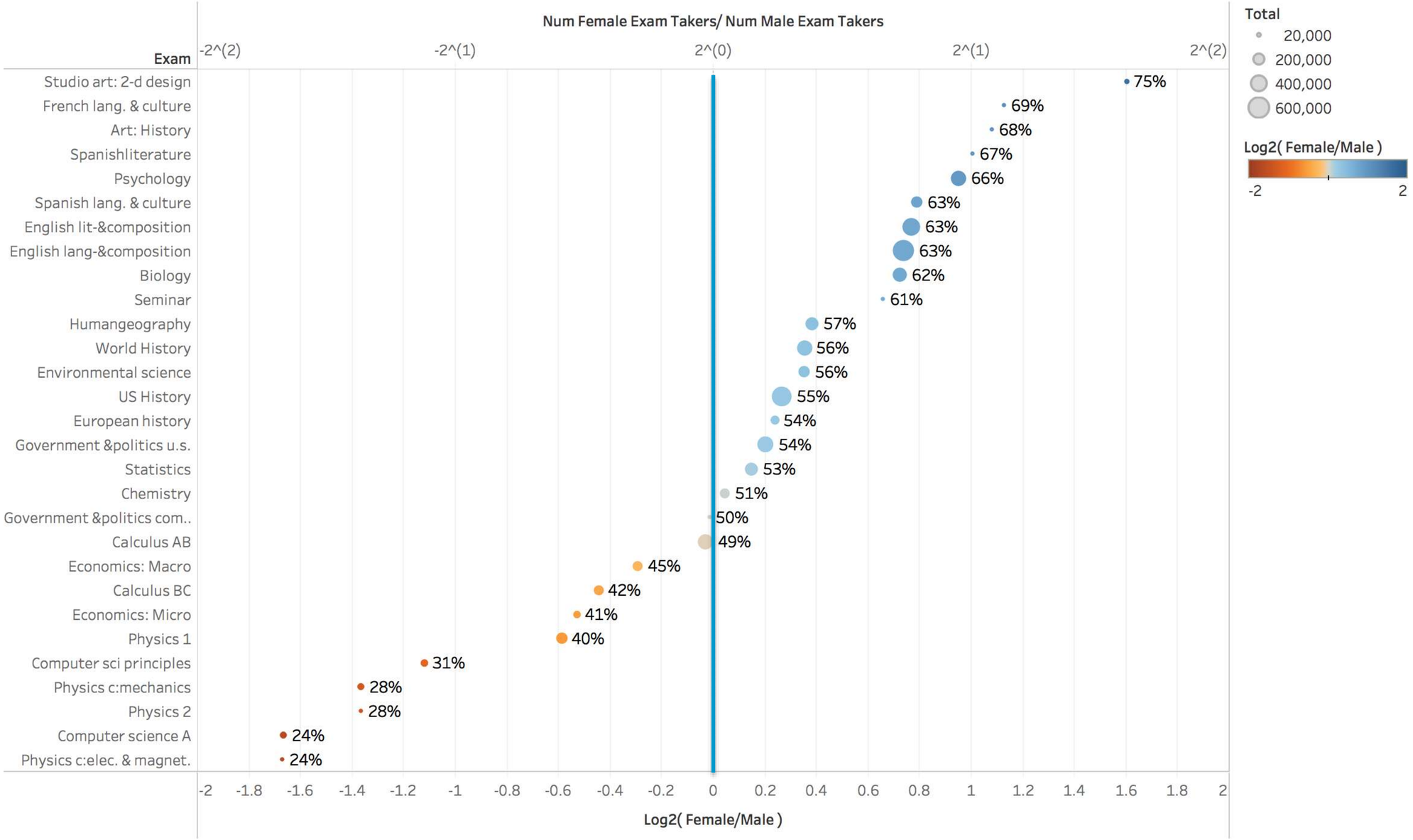
(43% have *never* offered CS.)

< 1% of Georgia students take CS

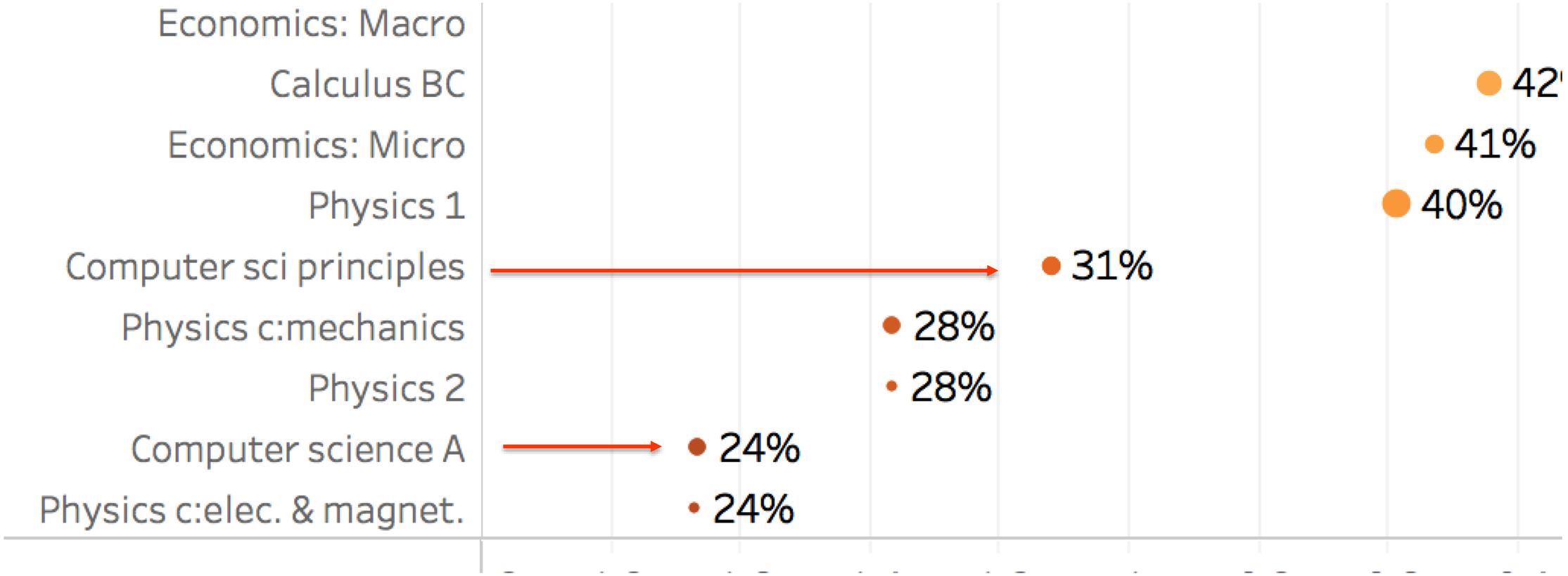
Thanks to Bryan Cox and Miranda Parker

Log-based Comparison of Number of Female and Male Exam Takers in 2018

For Advanced Placement exams taken by more than 20,000 individuals in total



Data and Visualization from Barbara Ericson and Willa Hua



Data and Visualization from Barbara Ericson and Willa Hua

Consider the scale

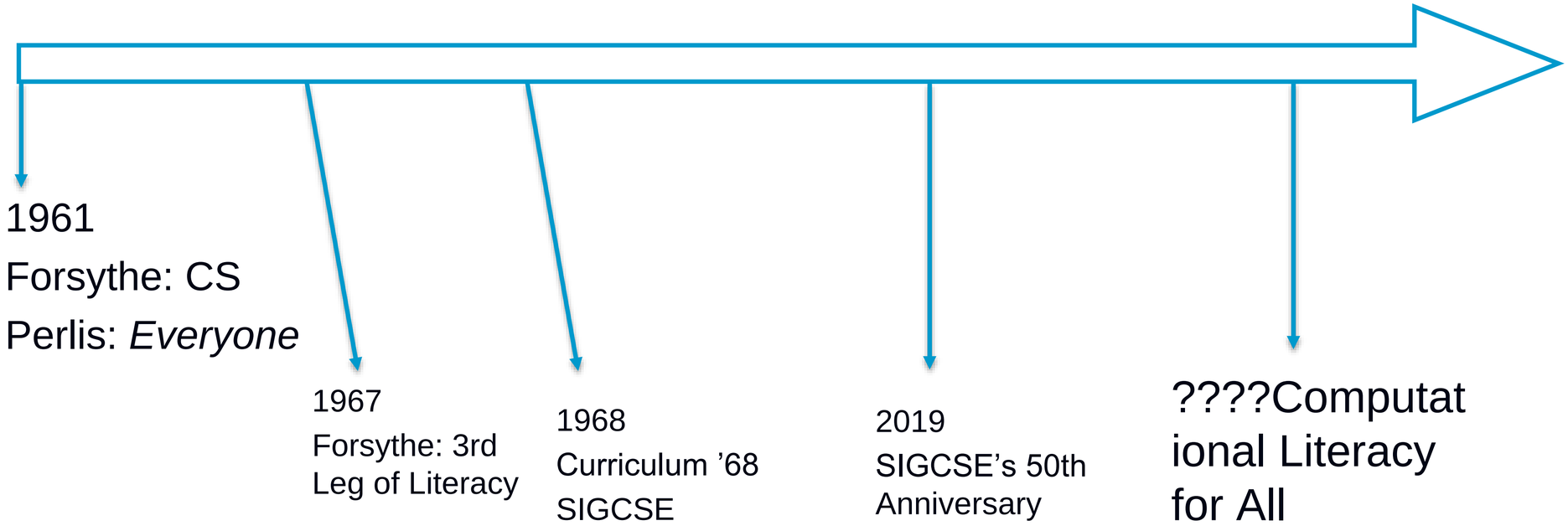
- AP CS A had 66K exam takers in 2018.
- AP CS Principles had 76K.

- AP English Lit has 580K
- AP Calculus has 305K

- There are 15.1 million high school students in US.

Perhaps 90% of US high school students *NEVER* see any CS

Our timeline



Developing computational literacy for all

BOOTSTRAP
Equity • Scale • Rigor

- Inspiring examples:
  - Bootstrap Algebra, Data Science, and Physics
  - Project GUTS
  - CT-STEM at Northwestern

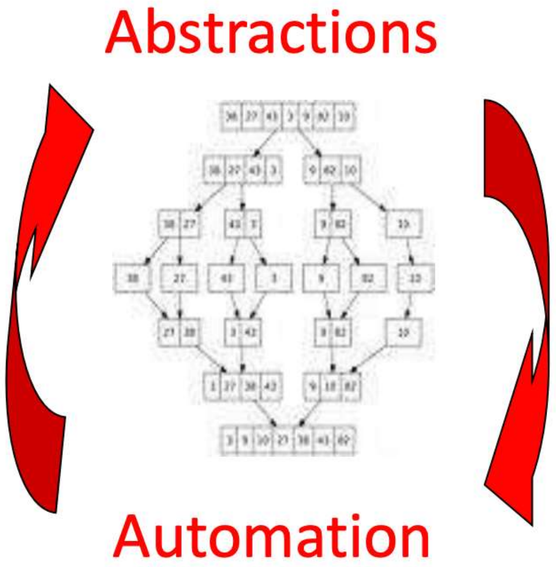
software carpentry

- End-user programming is growing
  - For every professional software developer in the US, there are 4-9 end-user programmers.

Computational Thinking

- Original definition from Seymour Papert, popularized by Jeannette Wing.

- Wing claimed that knowing about algorithms and computer architecture would influence everyday activities like picking a cashier line at the supermarket and packing your backpack.



Alternative: Computation for Thinking about Everything

Operational Definition of CT

- From ISTE and CSTA:

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

Computational Thinking

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

Specification of Problems

Use Automation

Organizing and Analyzing Data

Data representation

Use of models and simulations

Exploring a range of solutions

Efficiency and effectiveness in problem-solving

Engineering Thinking

ROYAL ACADEMY OF ENGINEERING

"Making 'things' that work and making 'things' work better."

Figure 1 - Centre for Real-World Learning engineering habits of mind	
Systems thinking	Seeing whole systems and parts and how they connect, pattern-sniffing, recognising interdependencies, synthesising
Problem-finding	Clarifying needs, checking existing solutions, investigating contexts, verifying
Visualising	Being able to move from abstract to concrete, manipulating materials, mental rehearsal of physical space and of practical design solutions
Improving	Relentlessly trying to make things better by experimenting, designing, sketching, guessing, conjecturing, thought-experimenting, prototyping
Creative problem-solving	Applying techniques from different traditions, generating ideas and solutions with others, generous but rigorous critiquing, seeing engineering as a 'team sport'
Adapting	Testing, analysing, reflecting, rethinking, changing both in a physical sense and mentally.

Engineering Thinking

ROYAL ACADEMY OF ENGINEERING

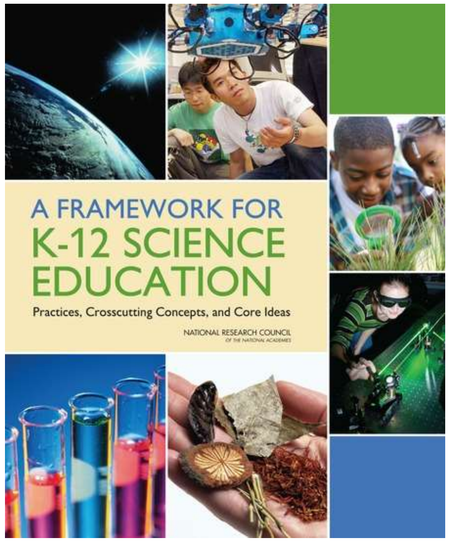
Figure 1 – Centre for Real-World Learning engineering habits of mind	
Systems thinking	Seeing whole systems and parts and how they connect, pattern-sniffing, recognising interdependencies, synthesising
Problem-finding	Clarifying needs, checking existing solutions, investigating contexts, verifying
Visualising	Being able to move from abstract to concrete, manipulating materials, mental rehearsal of physical space and of practical design solutions
Improving	Relentlessly trying to make things better by experimenting, designing, sketching, guessing, conjecturing, thought-experimenting, prototyping
Creative problem-solving	Applying techniques from different traditions, generating ideas and solutions with others, generous but rigorous critiquing, seeing engineering as a 'team sport'
Adapting	Testing, analysing, reflecting, rethinking, changing both in a physical sense and mentally.

- Specification of Problems
- Organizing and Analyzing Data
- Data representation
- Use of models and simulations
- Exploring a range of solutions
- Efficiency and effectiveness in problem-solving

Scientific Thinking

- From Krajcik and Merritt (2012)

  - Asking questions (for science) and defining problems (for engineering)
  - Developing and using models
  - Planning and carrying out investigations
  - Analyzing and interpreting data
  - Using mathematics, information and computer technology, and computational thinking
  - Constructing explanations (for science) and designing solutions (for engineering)
  - Engaging in argument from evidence
  - Obtaining, evaluating, and communicating information



Scientific Thinking

- Asking questions (for science) and defining problems (for engineering)
- Developing and using models
- Planning and carrying out investigations
- Analyzing and interpreting data
- Using mathematics, information and computer technology, and computational thinking
- Constructing explanations (for science) and designing solutions (for engineering)
- Engaging in argument from evidence
- Obtaining, evaluating, and communicating information

Specification of Problems

Organizing and Analyzing Data

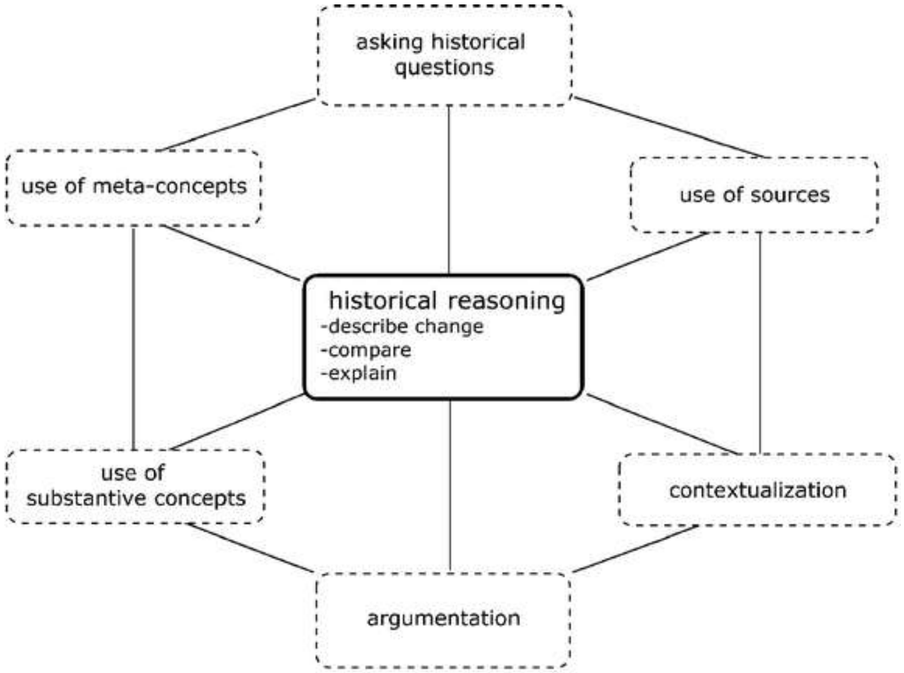
Data representation

Use of models and simulations

Exploring a range of solutions

Historical Thinking

- Determining a significant fact, problem, or question
- Interpret evidence
- Identify cause and consequence
- Use multiple perspectives

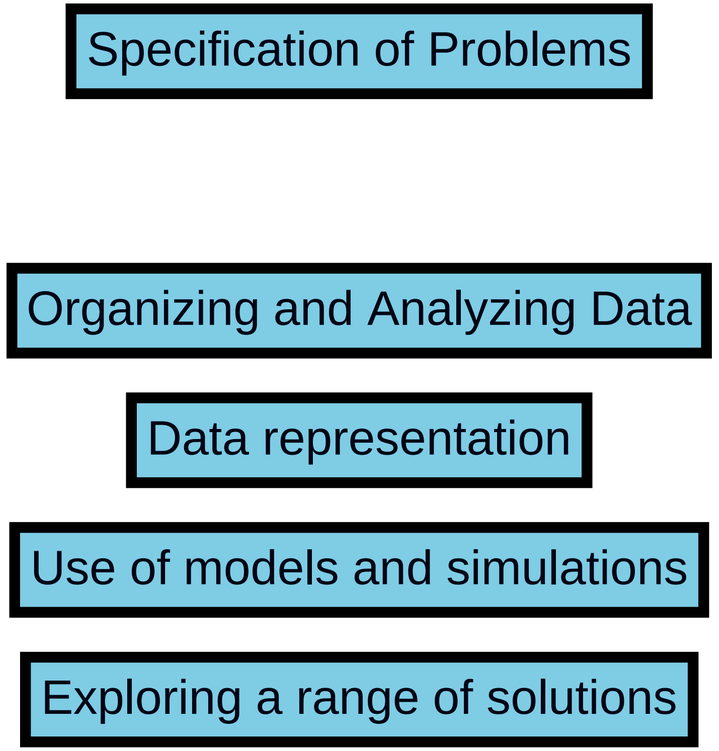


THE Historical Thinking PROJECT

Promoting critical historical literacy for the 21st century

Historical Thinking

- Determining a significant fact, problem, or question
- Interpret evidence
- Identify cause and consequence
- Use multiple perspectives

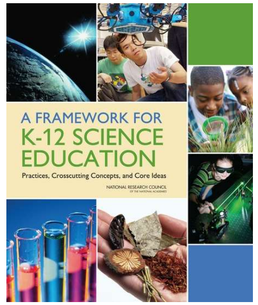


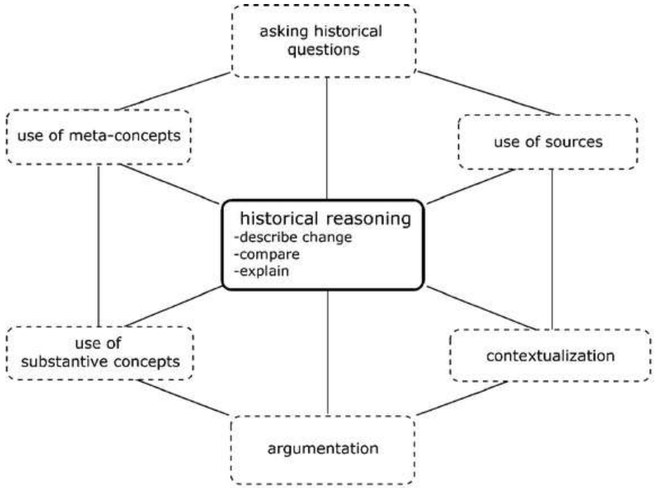
Computing is a medium for all of these

Figure 1 – Centre for Real-World Learning engineering habits of mind	
Systems thinking	Seeing whole systems and parts and how they connect, pattern-sniffing, recognising interdependencies, synthesising
Problem-finding	Clarifying needs, checking existing solutions, investigating contexts, verifying
Visualising	Being able to move from abstract to concrete, manipulating materials, mental rehearsal of physical space and of practical design solutions
Improving	Relentlessly trying to make things better by experimenting, designing, sketching, guessing, conjecturing, thought-experimenting, prototyping
Creative problem-solving	Applying techniques from different traditions, generating ideas and solutions with others, generous but rigorous critiquing, seeing engineering as a 'team sport'
Adapting	Testing, analysing, reflecting, rethinking, changing both in a physical sense and mentally.

ROYAL ACADEMY OF ENGINEERING

- Asking questions (for science) and defining problems (for engineering)
- Developing and using models
- Planning and carrying out investigations
- Analyzing and interpreting data
- Using mathematics, information and computer technology, and computational thinking
- Constructing explanations (for science) and designing solutions (for engineering)
- Engaging in argument from evidence
- Obtaining, evaluating, and communicating information

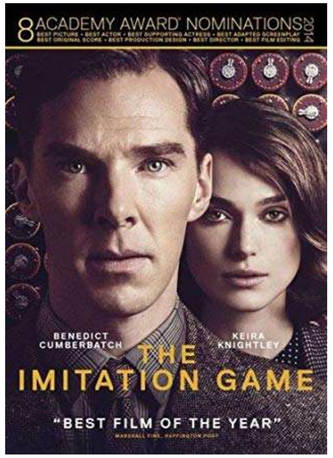




THE Historical Thinking PROJECT

Computing is a 21st Century Literacy.

Computing can be anything, and uniquely adds **automation** and **causal models** to help *learning.*



- Specification of Problems
- Use Automation
- Organizing and Analyzing Data
- Data representation
- Use of models and simulations
- Exploring a range of solutions
- Efficiency and effectiveness in problem-solving

“If we teach CS in other classes, can we teach enough?”

- How much of a foreign language do you need to communicate?
  To be fluent?
  Can’t you always learn more?

- We need novelists, journalists, and screenwriters.
  But the biggest impact of writing is the *everyday* use of it.

- Rich, Strickland, Binkowski, Moran, and Franklin (ICER 2017) asked the question:
  What’s the starting place for K-8 CS learners?

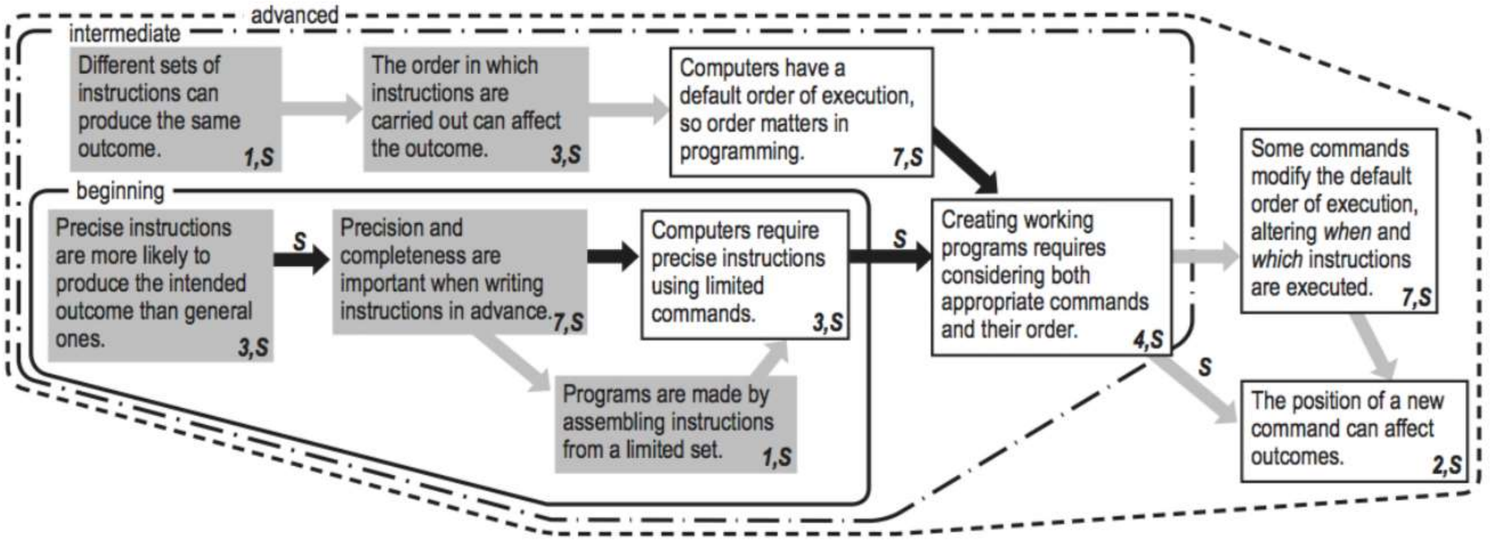


Figure 3: Sequence learning trajectory.

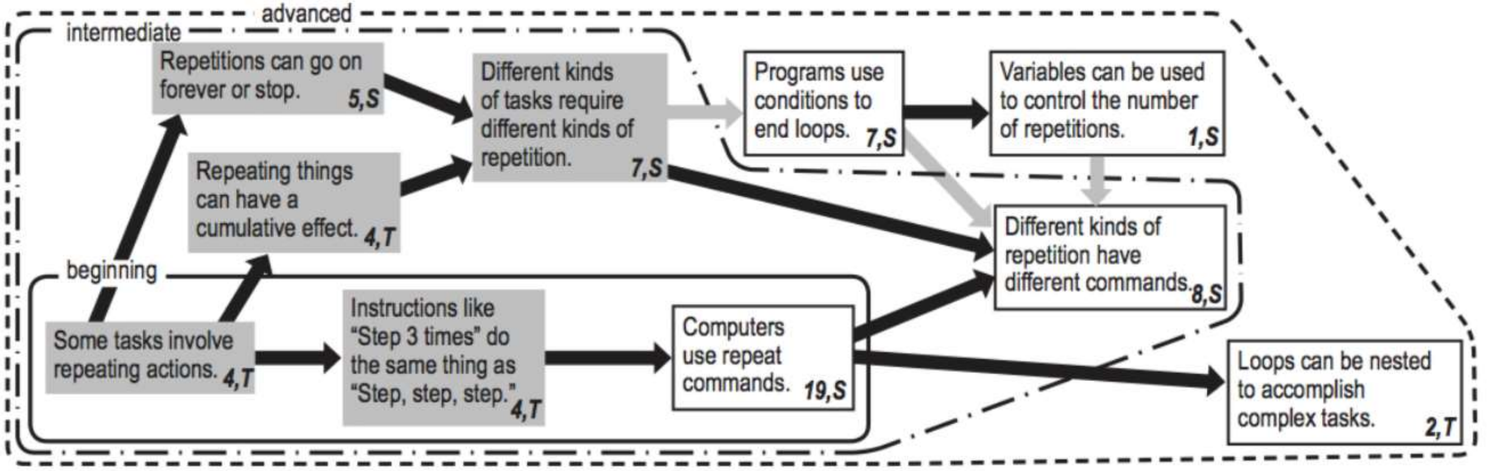


Figure 4: Repetition learning trajectory.

Proposed:
What comes first when learning programming?

1. Precision and completeness are important when writing instructions in advance.

2. Different sets of instructions can produce the same outcome.

3. Programs are made by assembling instructions from a limited set.

4. Some tasks involve repeating actions.

5. Programs use conditions to end loops.

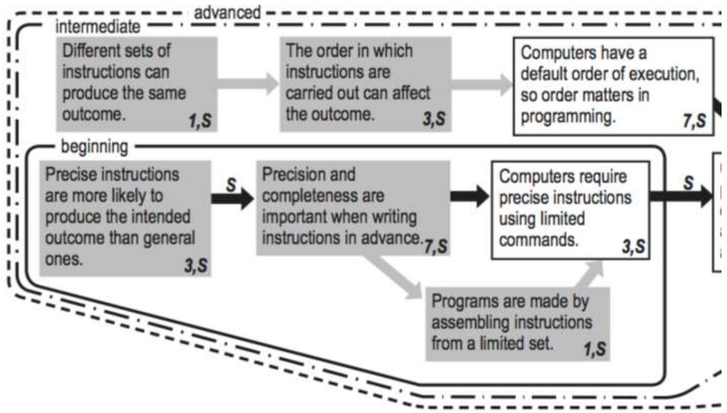
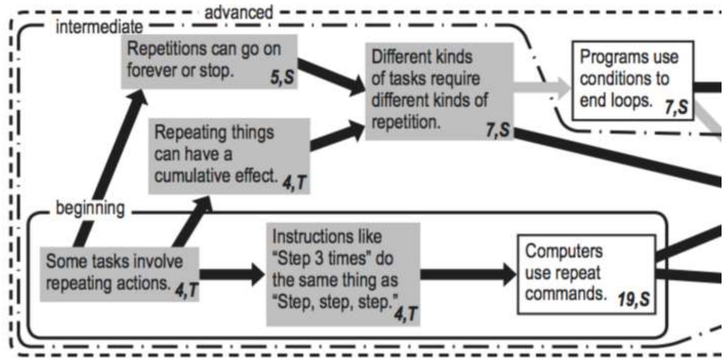


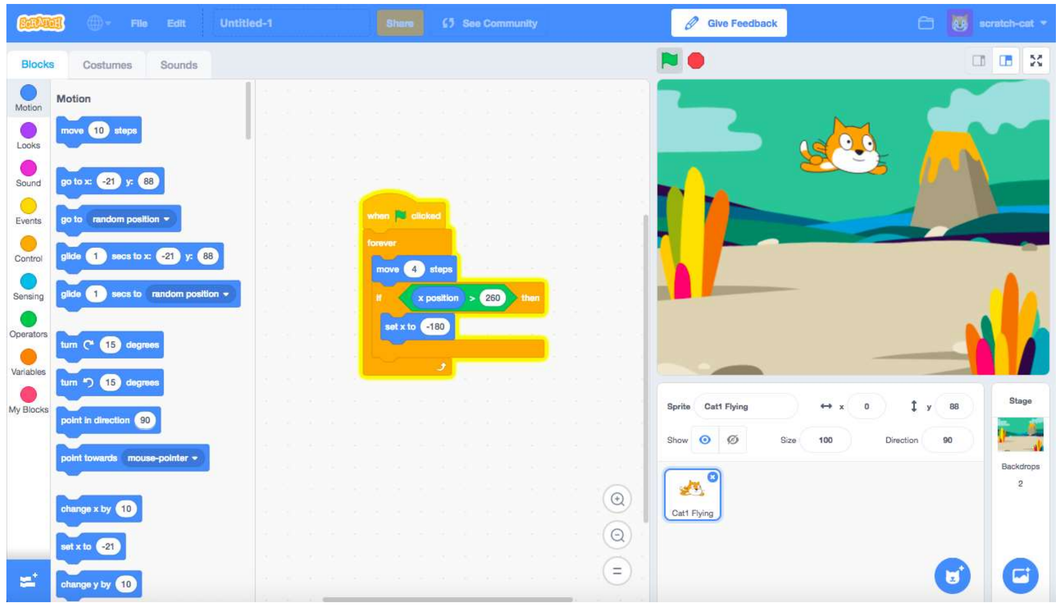
Figure 3: Sequence learning trajec



Scratch fluency doesn't need that whole list

- About 30 million users.
- Most Scratch projects are stories that use…
  - Only Forever loops
  - No booleans
  - Just movement and sequence.

There is *expressive* power in even a subset of CS.



Bootstrap:Algebra doesn't use all of that list

- Improves learning in algebra
- Students do not code repetition.
- Functional

BOOTSTRAP
Equity • Scale • Rigor

There is *learning* power in even a subset of CS.

Unit	Game Feature	Programming Concept	Math Concept
1	locating elements on screen	expressions, Circles of Evaluation	coordinates
2	creating text and images	string and image operations	domain, range, kinds of data
3–5	making moving images	defining functions, examples	multiple function representations: as formulas and as tables
6	determine when game elements are off-screen	Booleans and Boolean operators	inequalities
7	responding to key-presses	conditional	piecewise function
8	collision detection	(nothing new)	Pythagorean Theorem
9	polishing games for presentation	code reviews	explaining math concepts to others

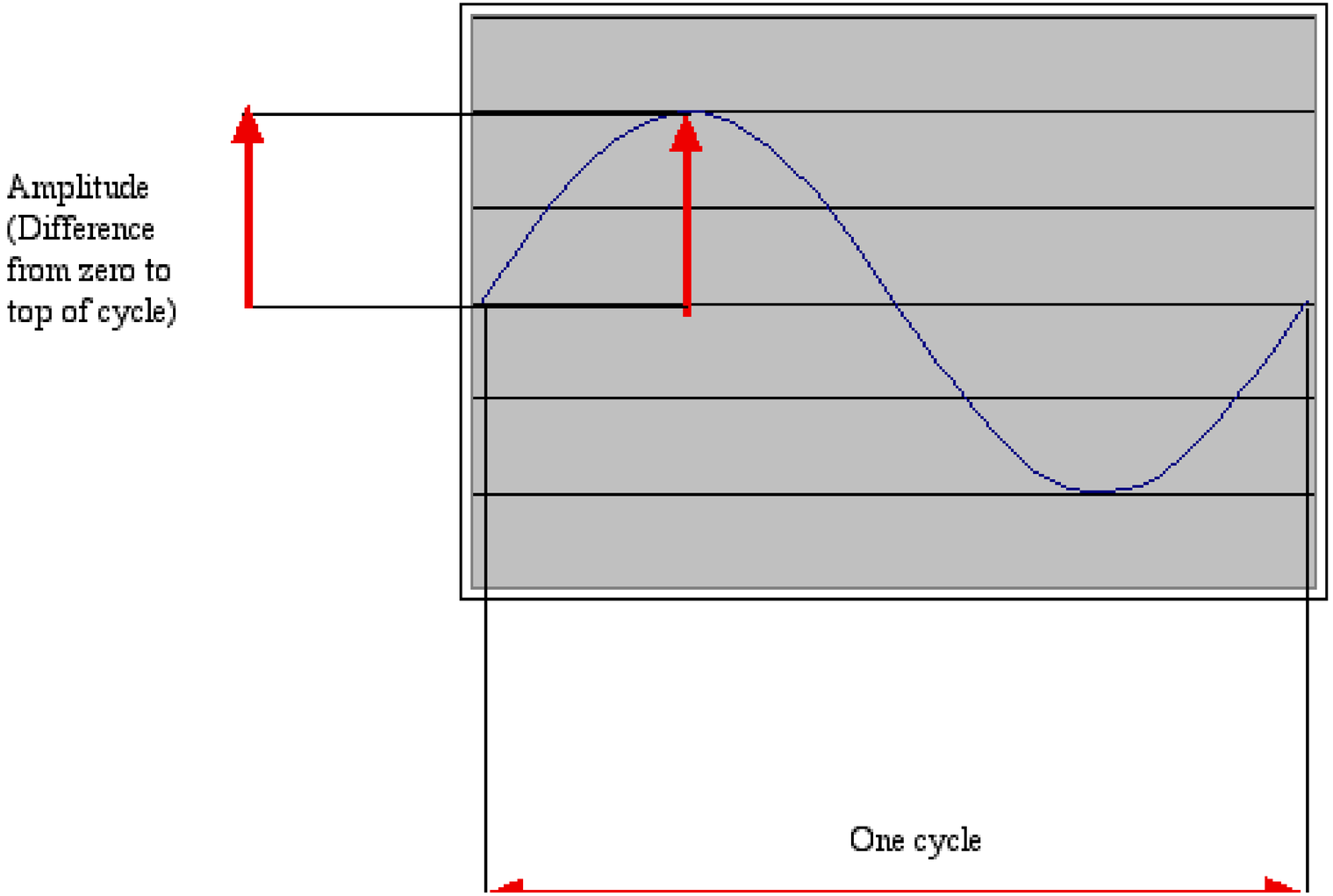
Figure 1: Curriculum structure: each unit introduces game, programming, and math concepts in parallel.

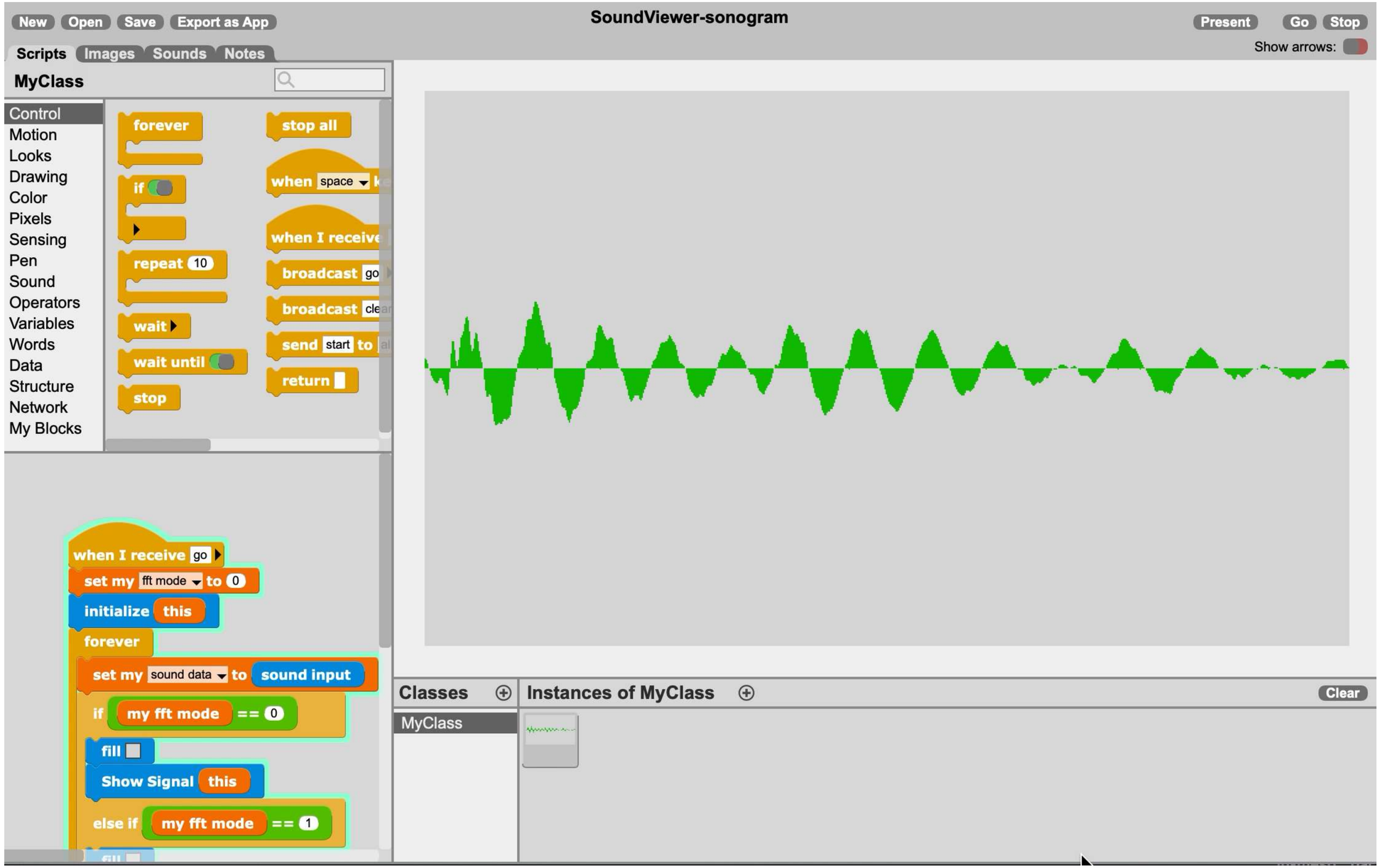
Schanzer, Fisler, Krishnamurthi, Felleisen, 2015

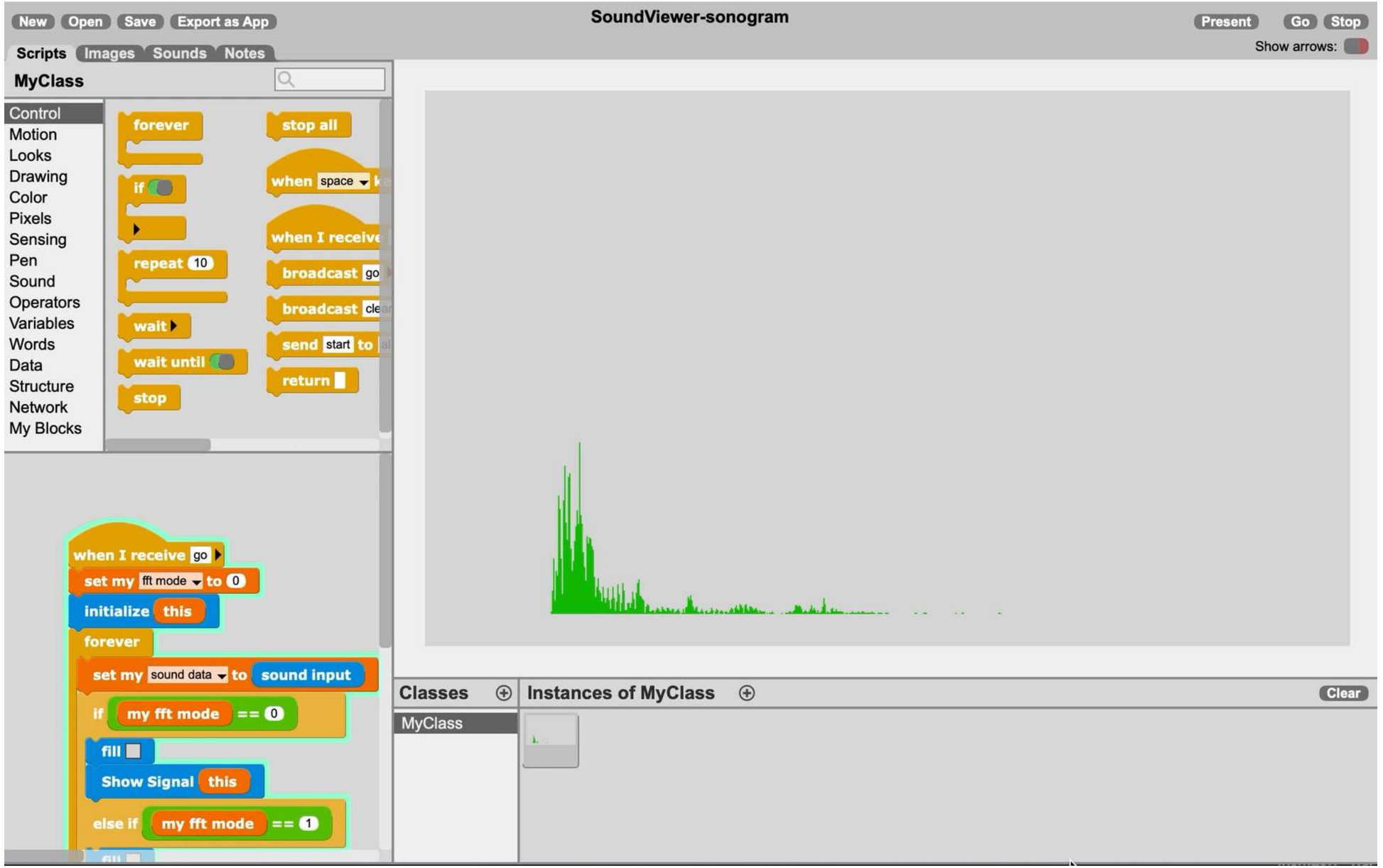
How would teaching for computational literacy look different?

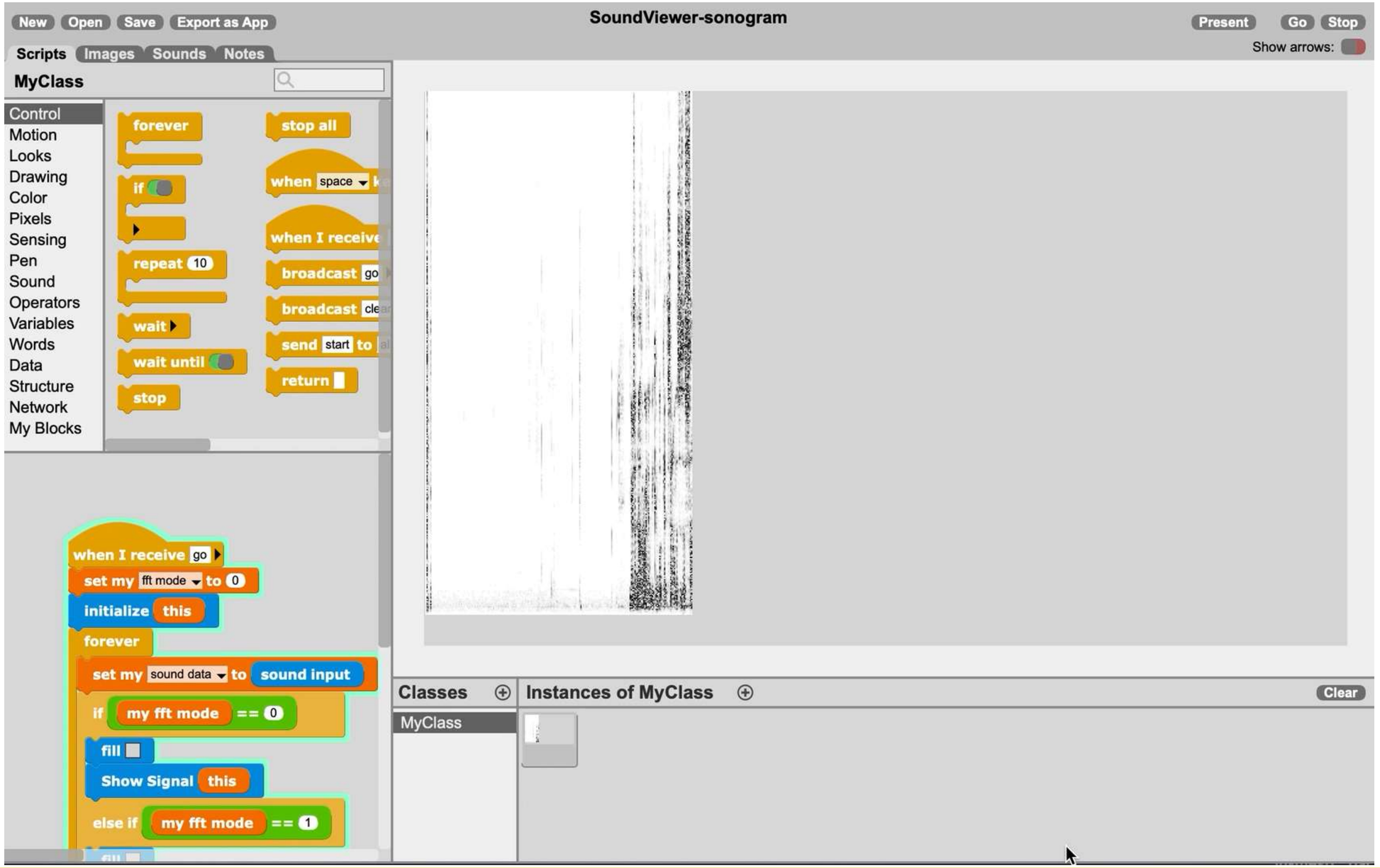
EXAMPLES

Example 1: How sound works: Acoustics, the physics of sound



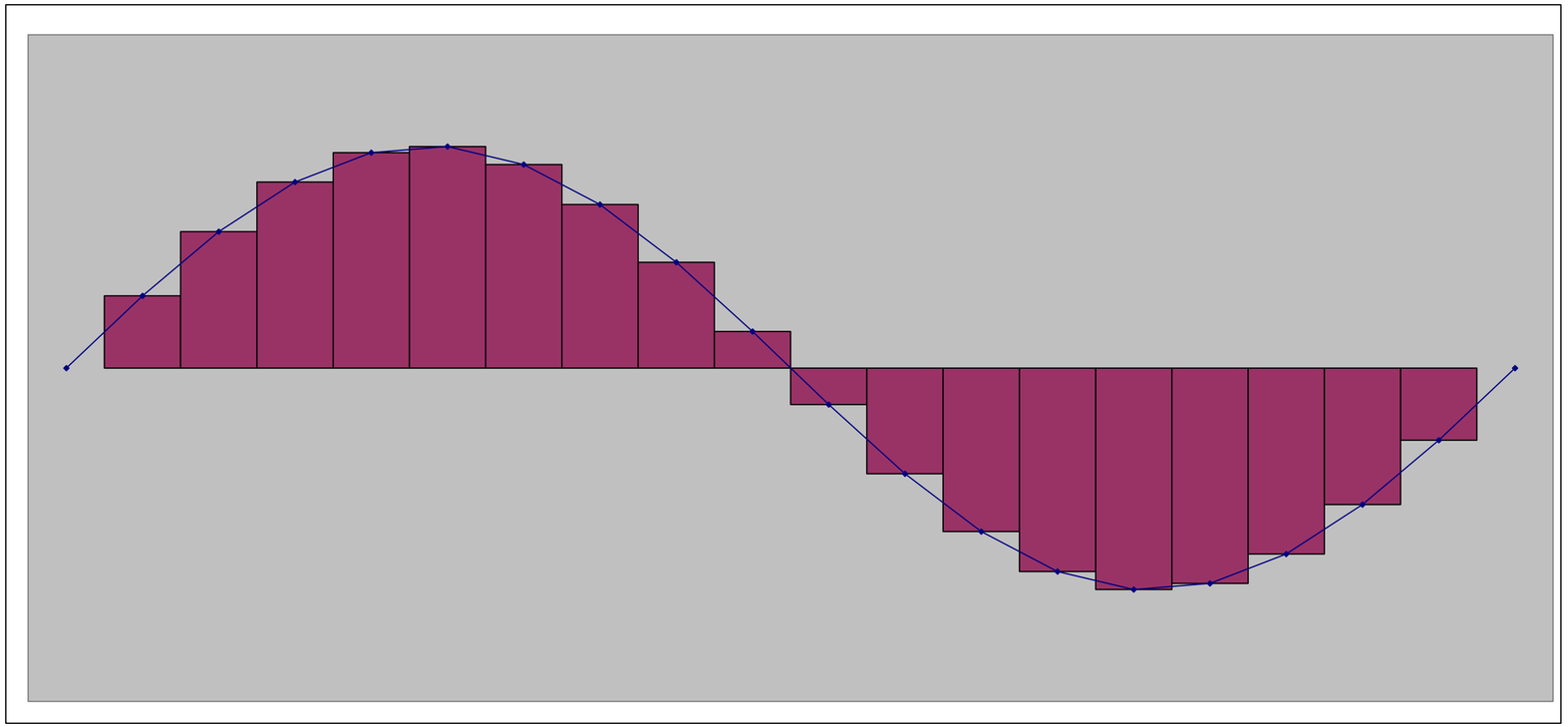

New
Open
Save
Export as App
SoundViewer-sonogram
Present
Go
Stop
Show arrows:
Scripts
Images
Sounds
Notes
MyClass
Control
Motion
Looks
Drawing
Color
Pixels
Sensing
Pen
Sound
Operators
Variables
Words
Data
Structure
Network
My Blocks
forever
if
repeat 10
wait
wait until
stop
stop all
when space
when I receive
broadcast go
broadcast
send start to
return
when I receive go
set my fft mode to 0
initialize this
forever
set my sound data to sound input
if my fft mode == 0
fill
Show Signal this
else if my fft mode == 1
Classes
Instances of MyClass
Clear
MyClass

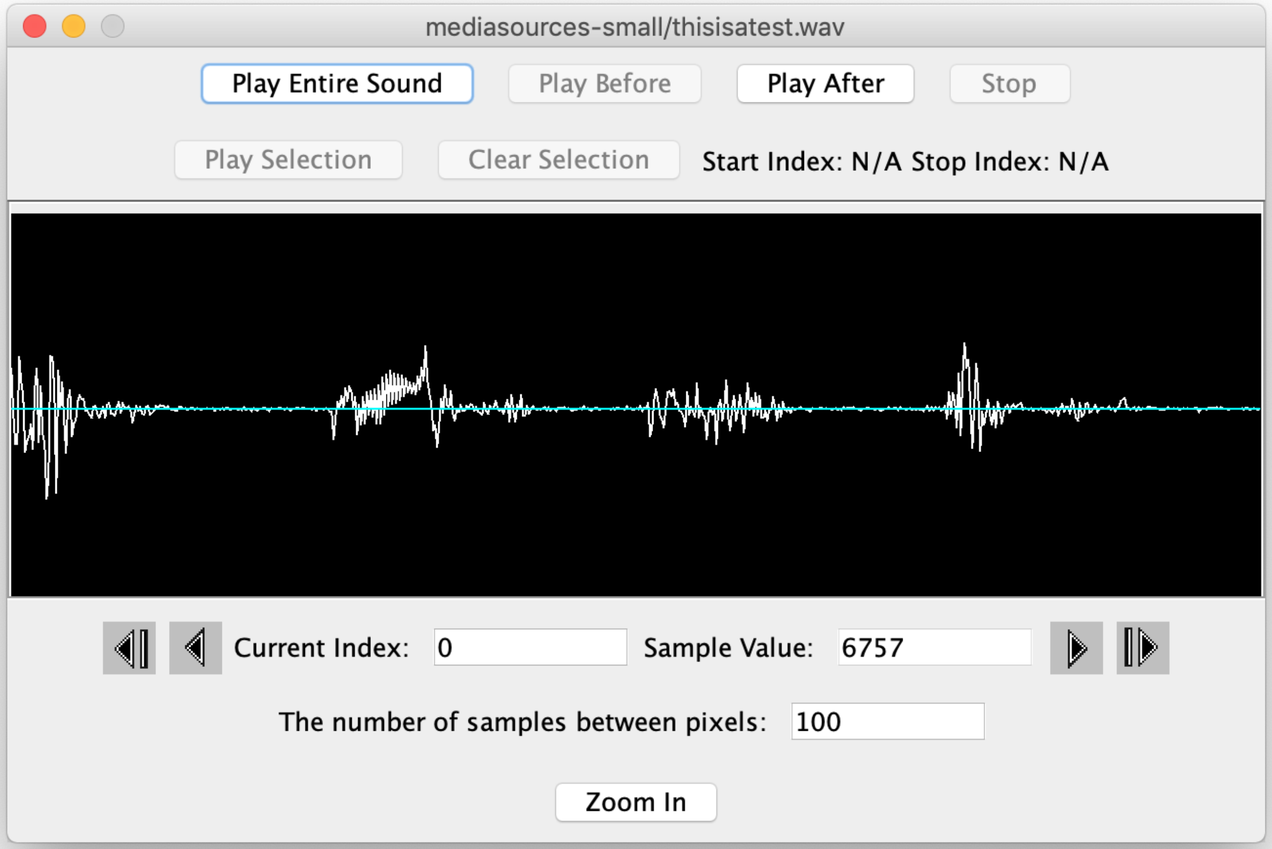

New
Open
Save
Export as App
SoundViewer-sonogram
Present
Go
Stop
Show arrows:
Scripts
Images
Sounds
Notes
MyClass
Control
Motion
Looks
Drawing
Color
Pixels
Sensing
Pen
Sound
Operators
Variables
Words
Data
Structure
Network
My Blocks
forever
if
repeat 10
wait
wait until
stop
stop all
when space
when I receive
broadcast go
broadcast
send start to
return
when I receive go
set my fft mode to 0
initialize this
forever
set my sound data to sound input
if my fft mode == 0
fill
Show Signal this
else if my fft mode == 1
Classes
MyClass
Instances of MyClass
Clear


New
Open
Save
Export as App
SoundViewer-sonogram
Present
Go
Stop
Show arrows:
Scripts
Images
Sounds
Notes
MyClass
Control
Motion
Looks
Drawing
Color
Pixels
Sensing
Pen
Sound
Operators
Variables
Words
Data
Structure
Network
My Blocks
forever
if
repeat 10
wait
wait until
stop
stop all
when space
when I receive
broadcast go
broadcast
send start to
return
when I receive go
set my fft mode to 0
initialize this
forever
set my sound data to sound input
if my fft mode == 0
fill
Show Signal this
else if my fft mode == 1
Classes
Instances of MyClass
Clear
MyClass

Digitizing Sound:
How do we get that into bytes?

- We can do the same to estimate the sound curve with *samples*.





```
>>>
>>> t = makeSound("thisisatest.wav")
>>> explore(t)
>>>
```

```
def increaseVolume(sound):
  for sample in getSamples(sound):
    value = getSampleValue(sample)
    setSampleValue(sample,value*4)
```

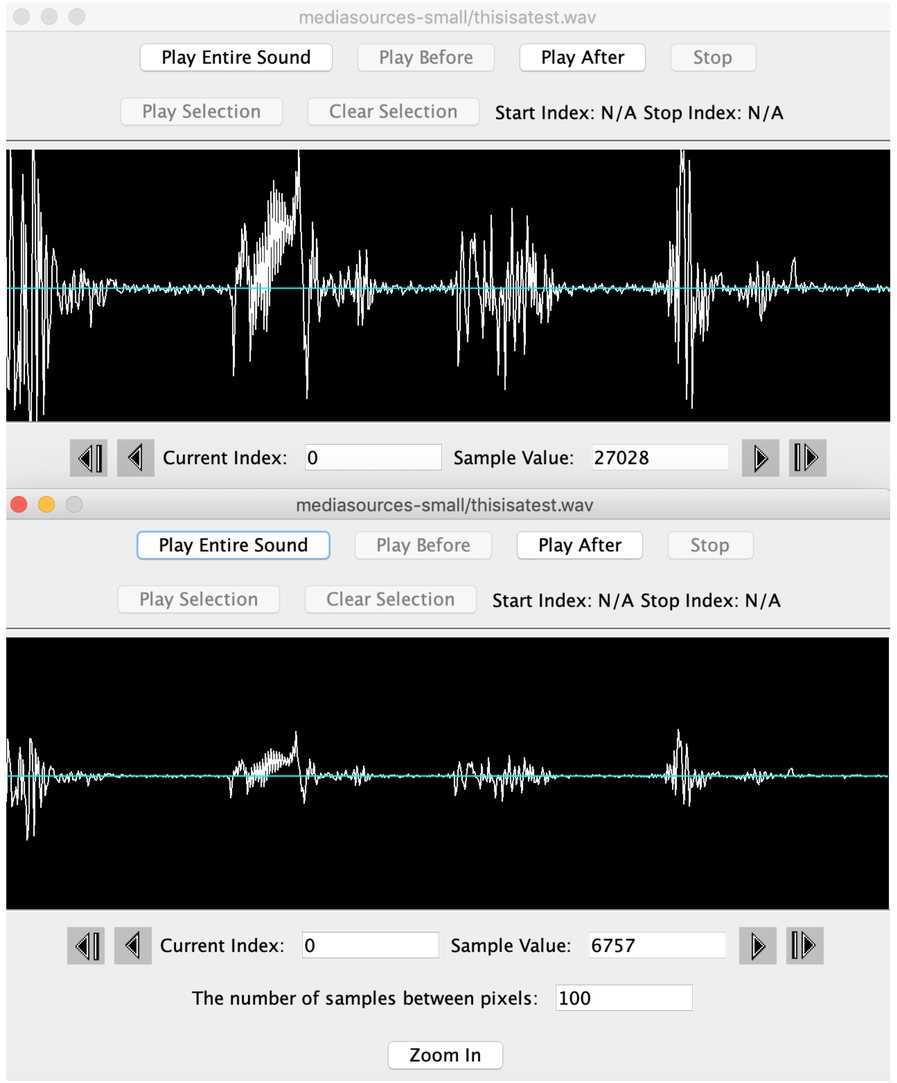
JES - demoCornellTech.py

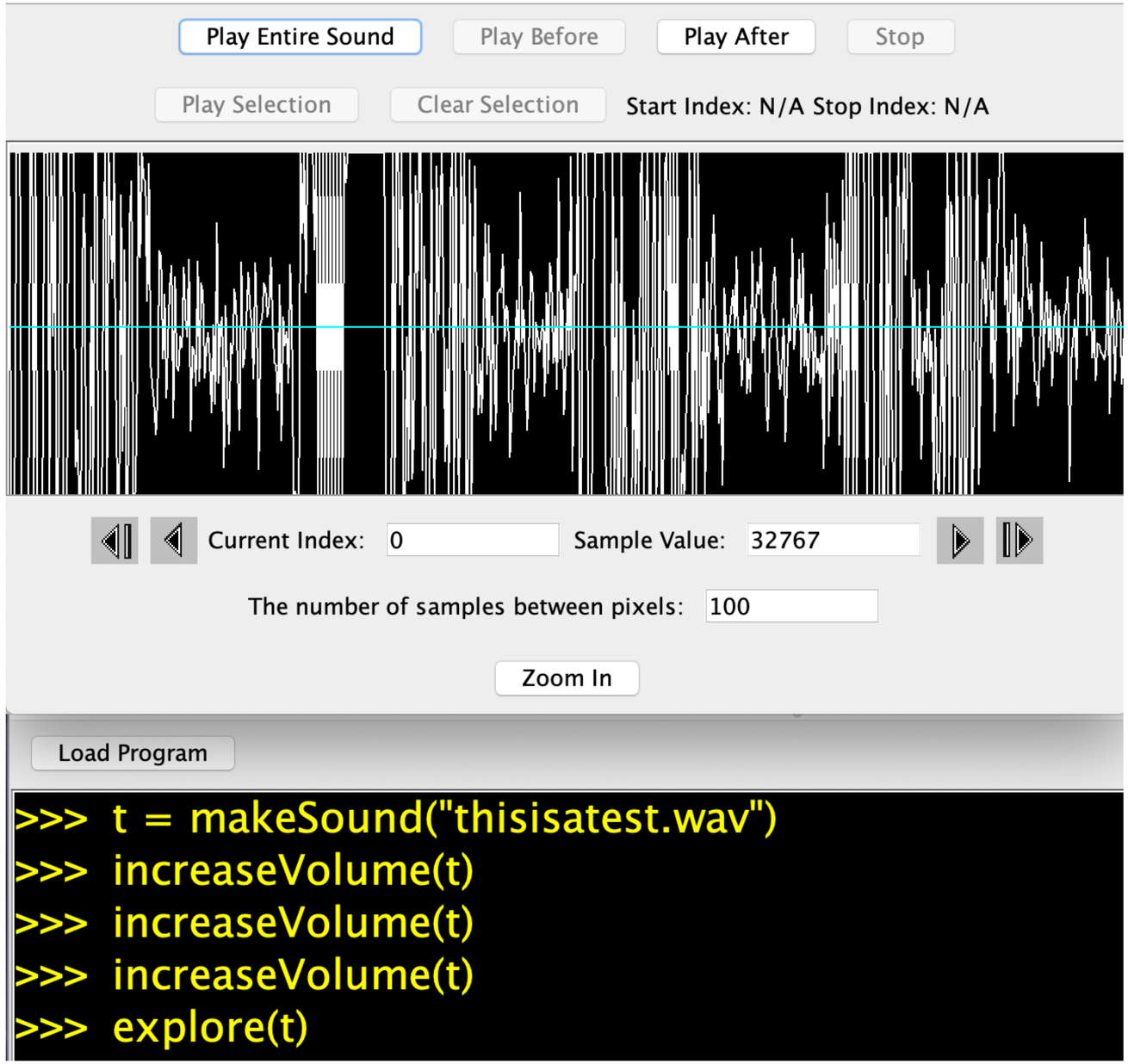
```
def increaseVolume(sound):
  for sample in getSamples(sound):
    value = getSampleValue(sample)
    setSampleValue(sample, value * 4)
```

Load Program | Watcher | Stop

```
>>> increaseVolume(t)
>>> explore(t)
>>>
>>>
>>>
```

For help on a particular JES function, move the cursor over it | Explain <click> | Line Number:1 Position: 1



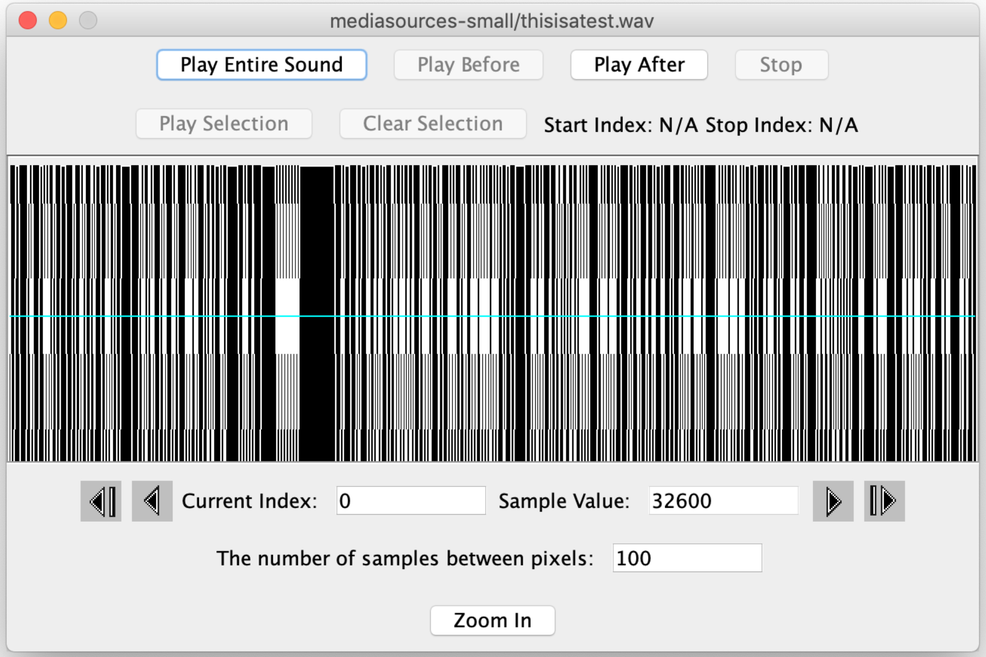

Play Entire Sound
Play Before
Play After
Stop
Play Selection
Clear Selection
Start Index: N/A Stop Index: N/A
Current Index:
0
Sample Value:
32767
The number of samples between pixels:
100
Zoom In
Load Program
>>> t = makeSound("thisisatest.wav")
>>> increaseVolume(t)
>>> increaseVolume(t)
>>> increaseVolume(t)
>>> explore(t)

```
def maximize(sound):
  for sample in getSamples(sound):
    value = getSampleValue(sample)
    if value >= 0:
      setSampleValue(sample, 32767)
    if value < 0:
      setSampleValue(sample, -32767)
```

```
16
17 def maximize(sound):
18   for sample in getSamples(sound):
19     value = getSampleValue(sample)
20     if value >= 0:
21       setSampleValue(sample,32600)
22     if value < 0:
23       setSampleValue(sample, -32600)
24
```

Load Program

```
>>> explore(t)
>>> t = makeSound("thisisatest.wav")
>>> maximize(t)
>>> explore(t)
```



Reflection

- Prediction
- 7 Lines
- 1 Bit
- Learning without writing a Program

```
def maximize(sound):
  for sample in getSamples(sound):
    value = getSampleValue(sample)
    if value >= 0:
      setSampleValue(sample, 32767)
    if value < 0:
      setSampleValue(sample, -32767)
```

**Teaching CS for insight into our world,
not software development.**

Example 2:
Subgoal Labeling

- Students are often overwhelmed when programming.

  "You've taught me so many details,
  I don't know which ones to use."

  (Clancy & Linn, 1990)

- How do we convey how to think about the *purpose* for the parts of the program? About *why* each part is there?

- Richard Catrambone (1994) invented a way to label the *subgoals* in examples provided to students.

Example of Written Materials

With Subgoals

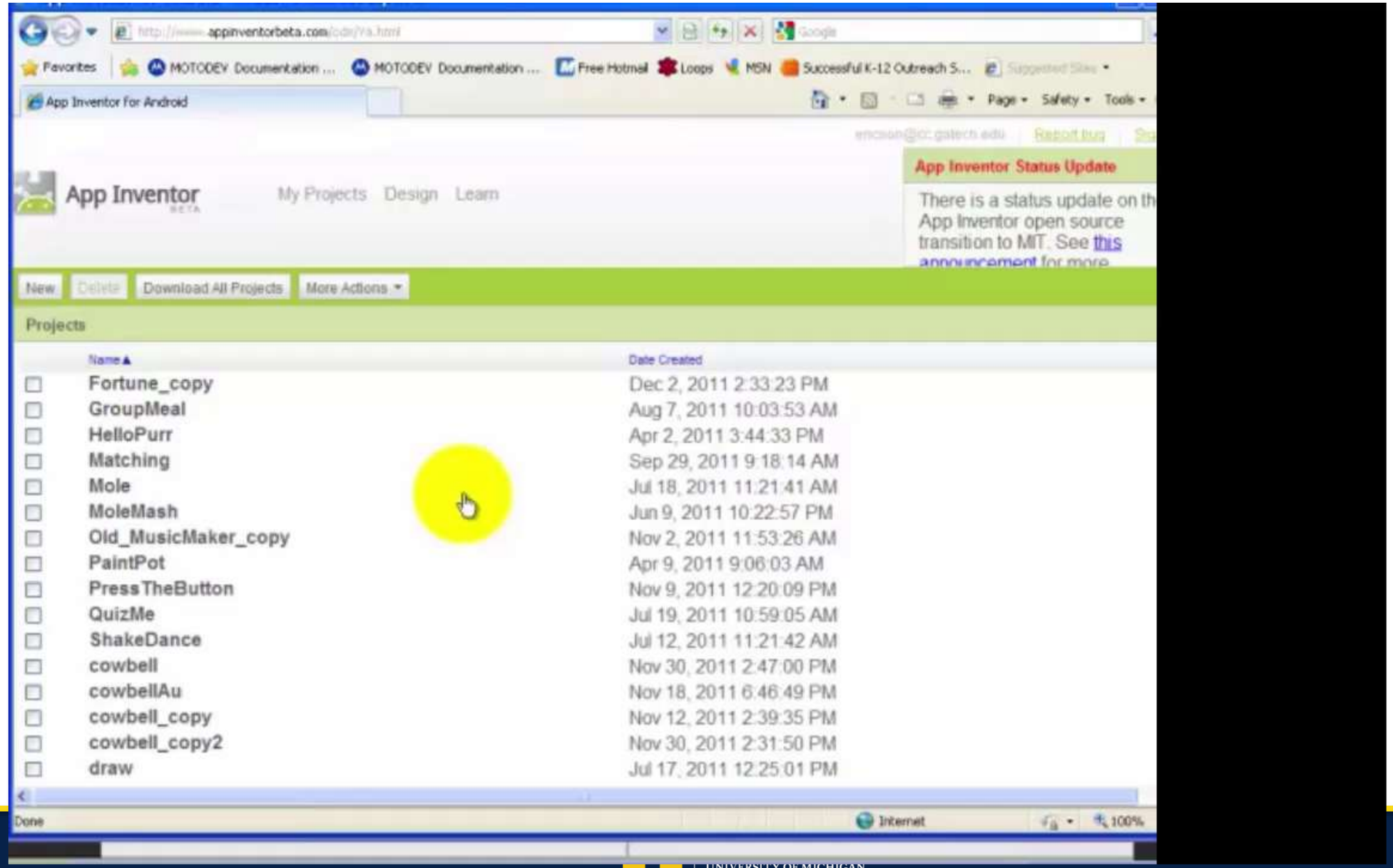
- **Define Variables from Built-in**
- Click on "Built-In" and "Definition" and pull out a def variable.
- Click on the "variable" and replace it with "fortuneList". This creates a variable called "fortuneList".
- Click on "Lists" and drag out a call make a list
- Click on "Text" and drag out a text text block and drop it next to "item". Click on the rightmost "text" and replace it with your first fortune.
- **Handle Events from My Blocks**
- Click on "My Blocks" and "Button1".
- Drag out a when Button1.Click.

Without Subgoals

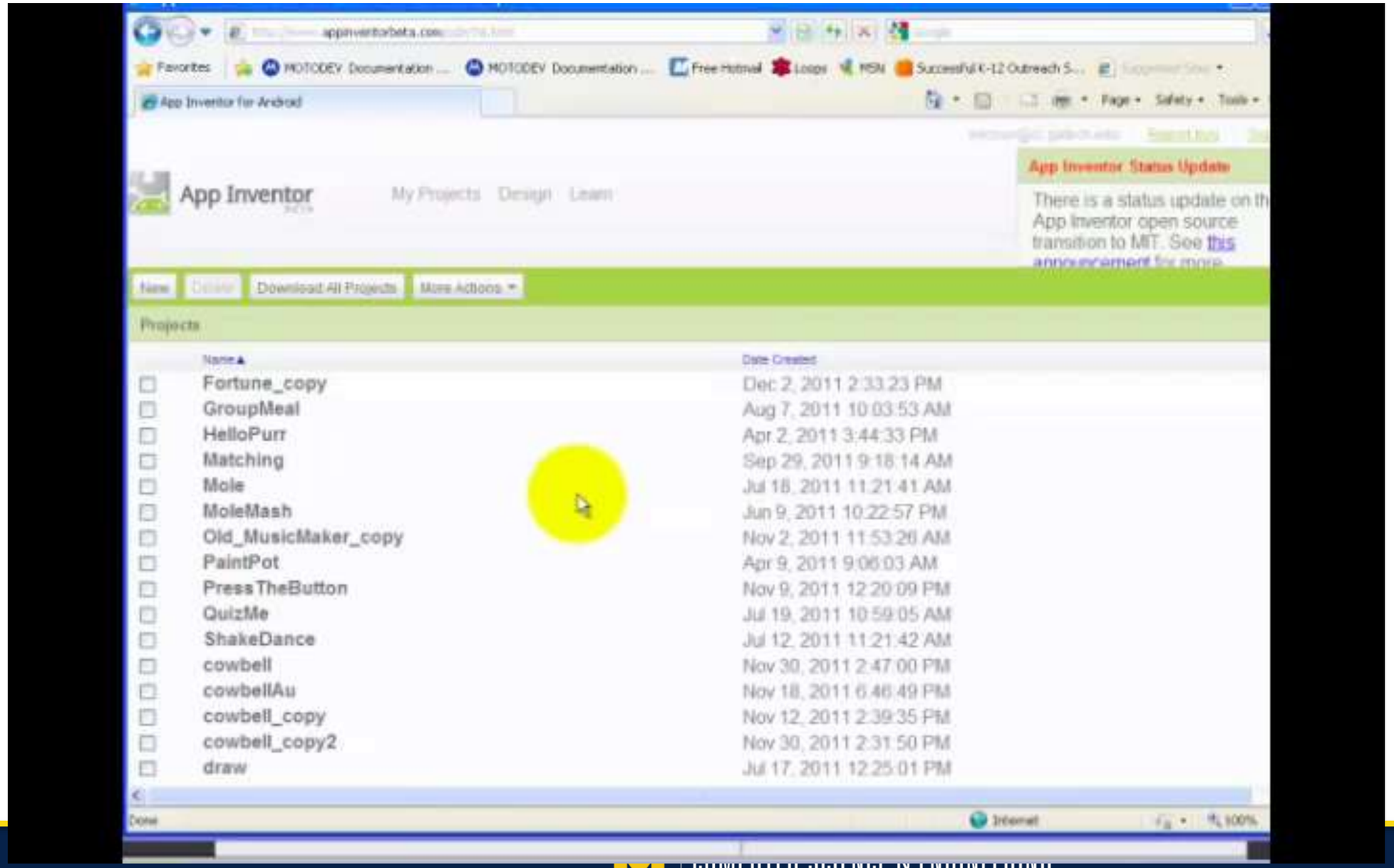
- Click on "Built-In" and "Definition" and pull out a def variable.
- Click on the "variable" and replace it with "fortuneList". This creates a variable called "fortuneList".
- Click on "Lists" and drag out a call make a list
- Click on "Text" and drag out a text text block and drop it next to "item".
- Click on the rightmost "text" and replace it with your first fortune.
- Click on "My Blocks" and "Button1".
- Drag out a when Button1.Click.

Lauren Margelieux, Mark Guzdial, and Richard Catrambone, ICER 2012

Original Video



With Subgoals



Experiment with App Inventor

- Used subgoal labeling to teach Android App Inventor (a blocks-based programming environment)
- Two groups of undergraduate students:
  - One group was shown a video for how to use the software to build an App and given text listing the steps in the instruction.
  - Another group was given the video and the steps with subgoal labels.

Steps in Experiment

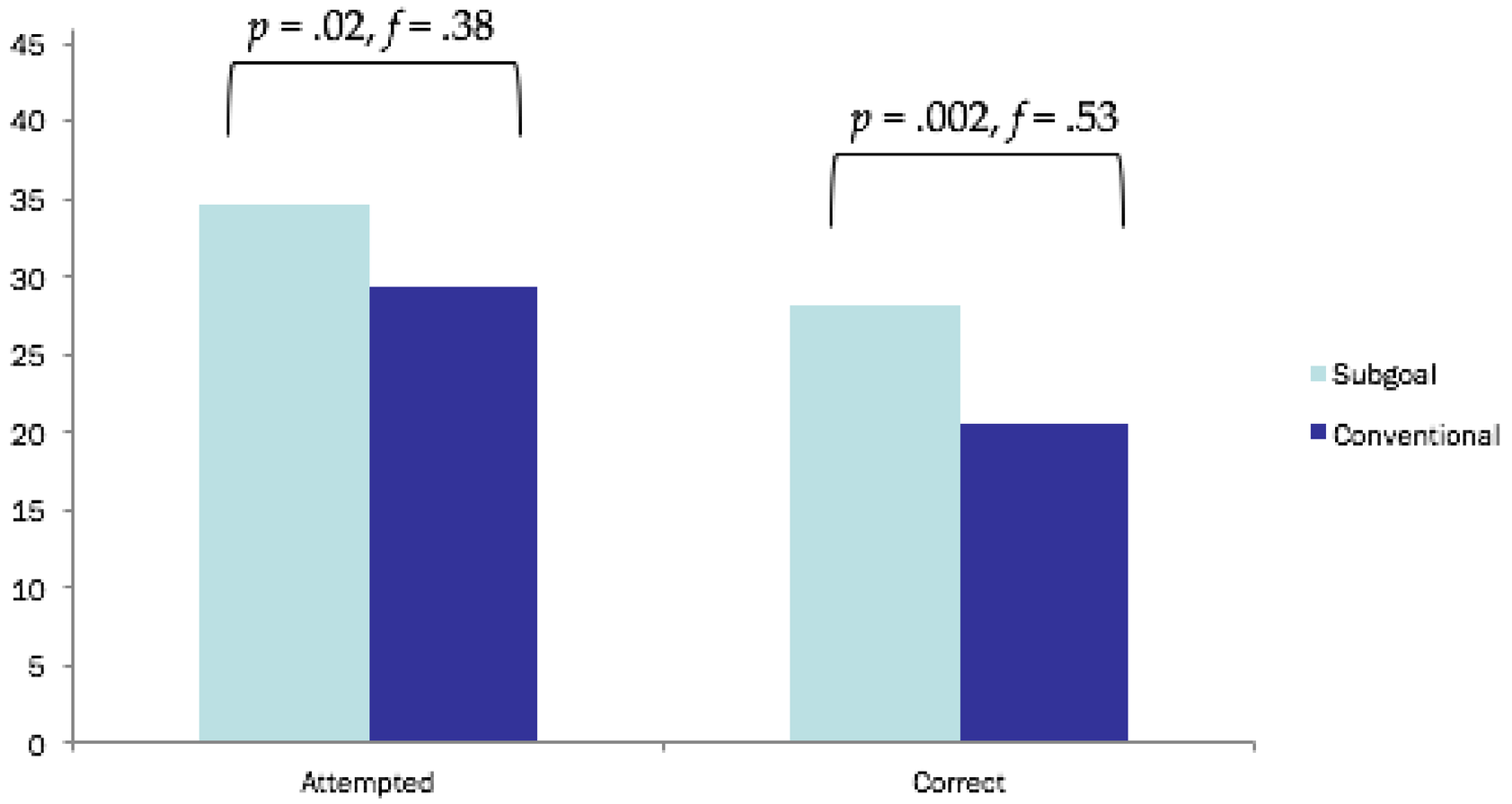
Week 1:

- Watch the video.
- Take a test to demonstrate ***understanding***.

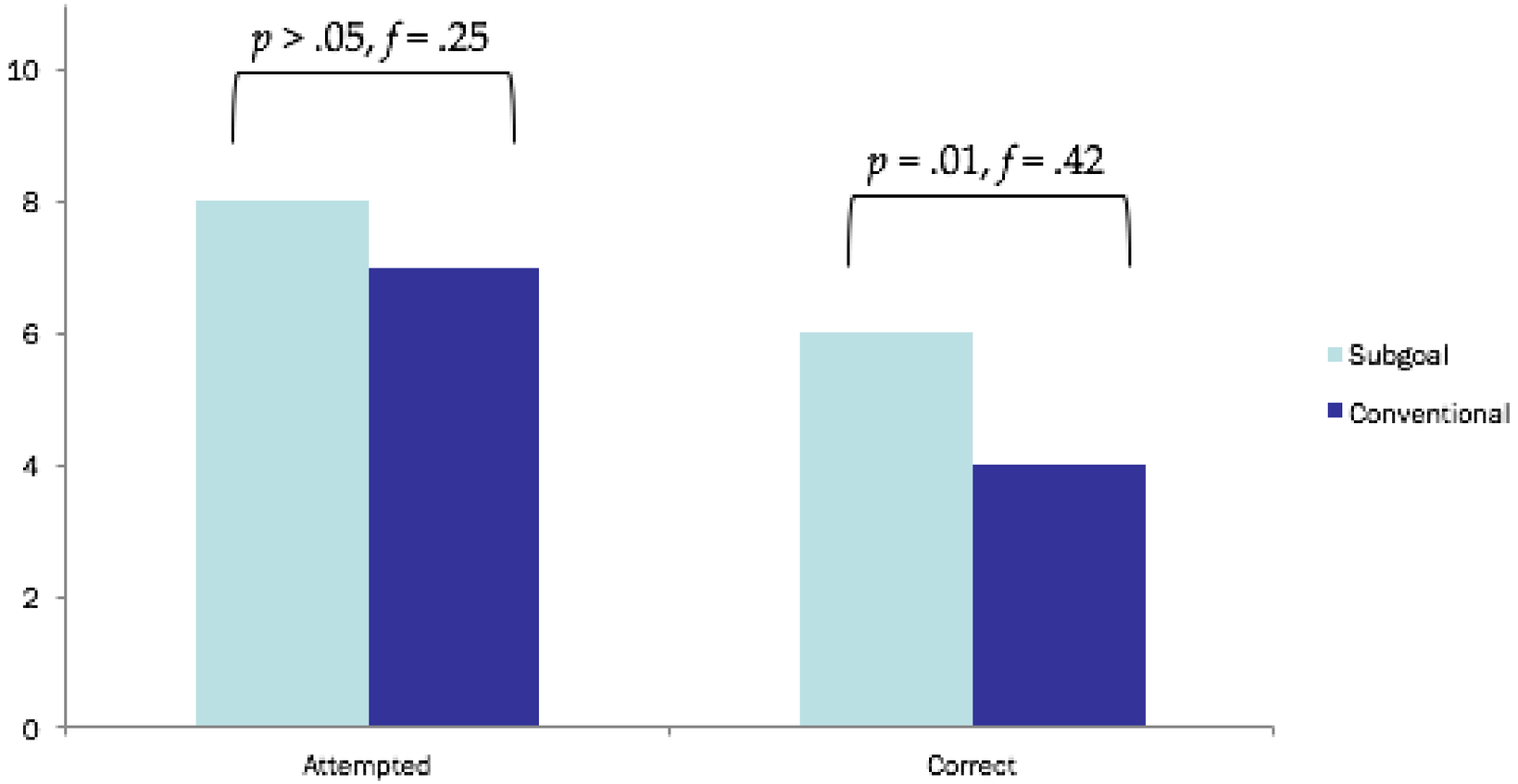
Week 2:

- Take a test to demonstrate ***retention***.
- Watch a new video.
- Take a test to demonstrate understanding of second video.
- Take a test where students must build a new app, ***transferring*** knowledge.

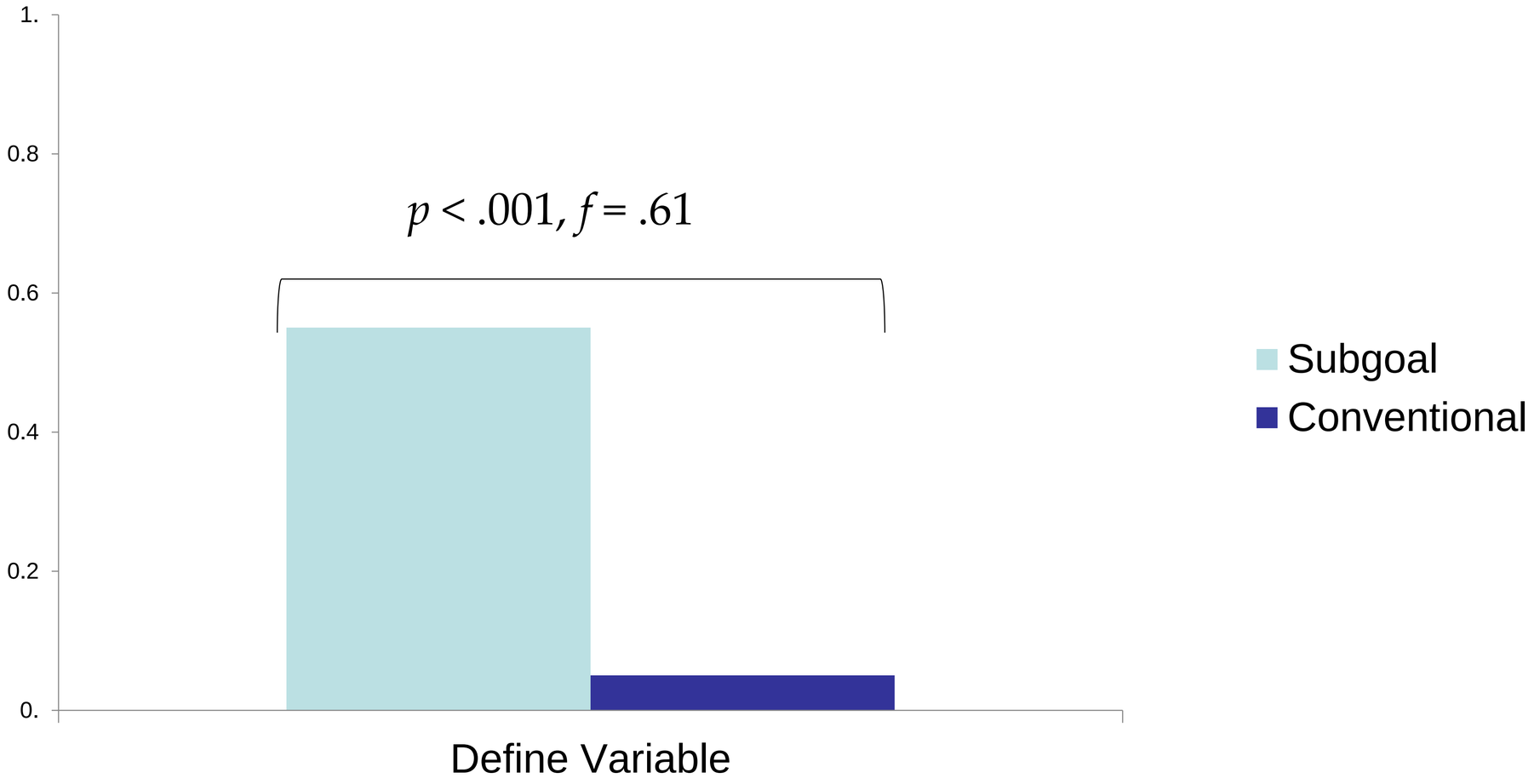
Results: Understanding



Results: Retention



Results: Define Variable Step in Transfer Task



It works, and in other settings

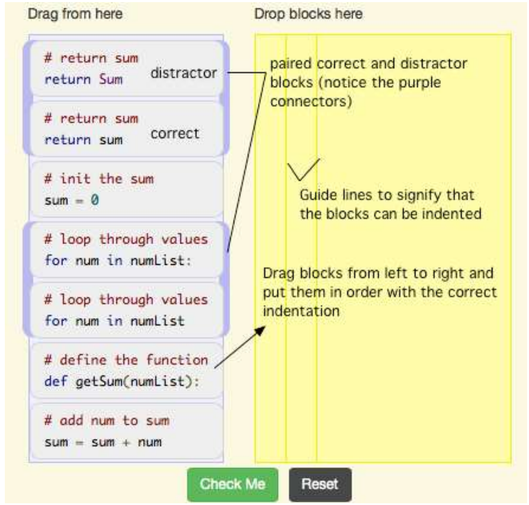
- Small changes that result in better understanding, retention, and transfer.
- Effect is twice as strong for high school teachers.
- Works in text-based programming languages, too

Improving the efficiency and effectiveness of Computing Education

Work by Lauren Margelieux and Briana Morrison.

We can make Computing Education better

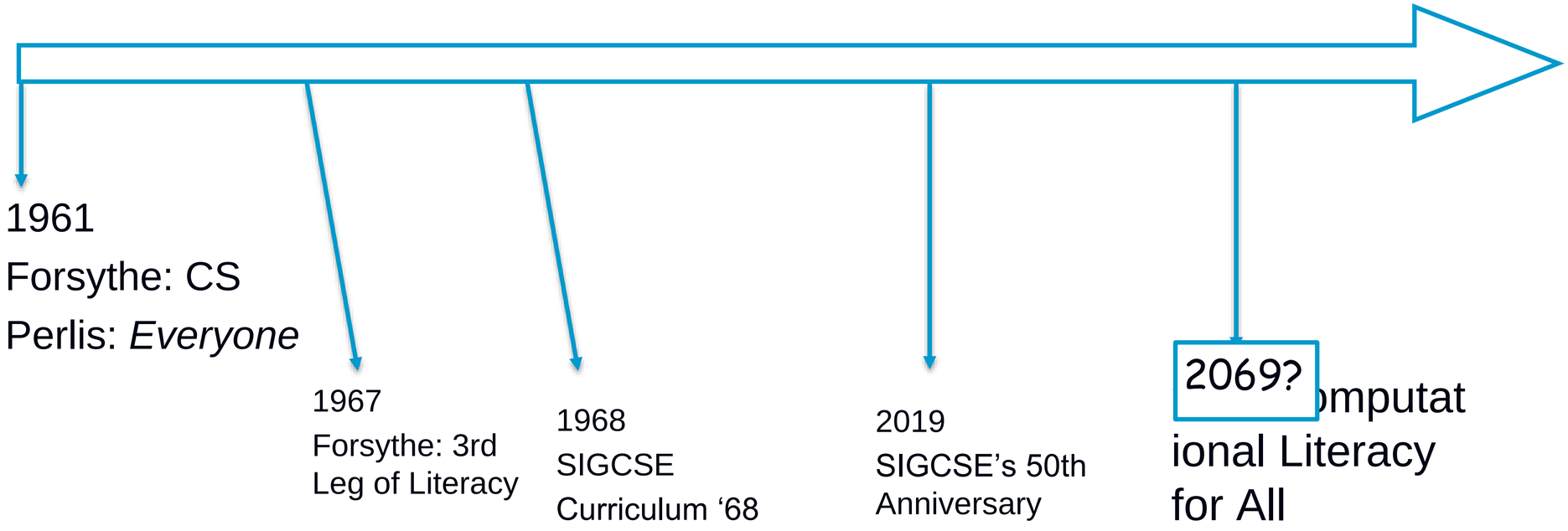
- Parsons Problems
- Worked Examples
- Pair Programming
- Media Computation



We will have to make learning computing more effective and efficient to reach *everyone*

Parsons Problem from Barbara Ericson; Pair programming picture from Natalie Freed

Our timeline



Call to Action

- Find our allies and grow the community.
- Invent, mutate, evolve.

Our Allies

- Computing across the curriculum…
  Allies across the curriculum.

- Learn from everyone we can:
  Physics and mathematics education research, educational psychology, learning sciences.

- We can’t be the experts in everything.
  We need others to tell us how to make computing **be** everything.

Special Interest Group on Computer Science Education

Our Mission: To provide a <u>global forum</u> for **<u>educators</u>** to discuss research and practice related to the learning and teaching of computing…at all education levels

Individuals → Allied Communities

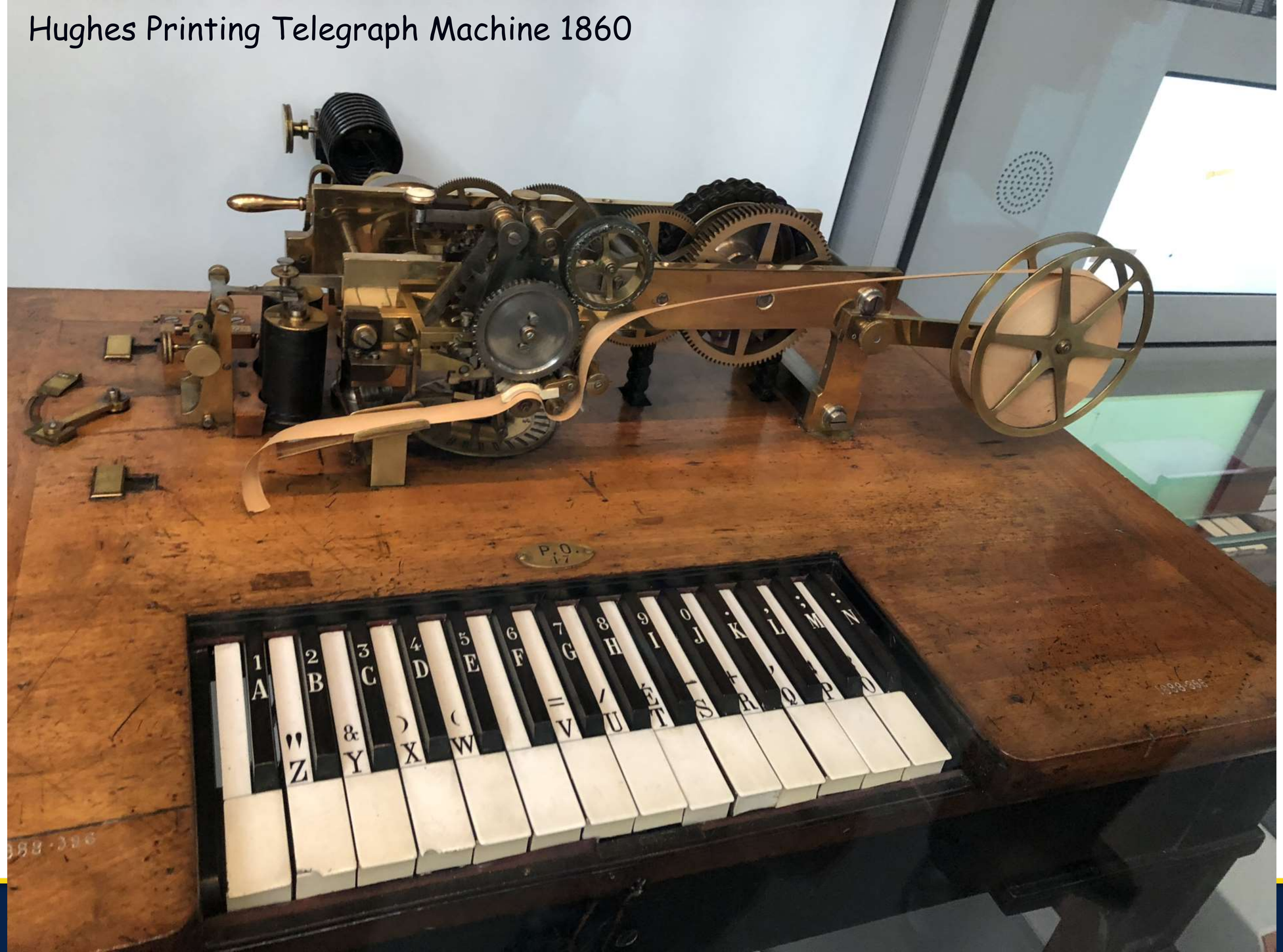
- Computer scientists in Schools of Education



- Special needs educators in Computer Science

- Math, science, engineering, and social studies teachers who teach with programming


Hughes Printing Telegraph Machine 1860



For 30 years, this was the common keyboard

- We may still be waiting for our QWERTY keyboard.
- How much better would we all be if we had adopted something even better than QWERTY?

We need to find what makes the great ideas of computing accessible.

Invent, evolve, mutate

- Invent:
  - New tools, languages, curricula, connections.
- But don’t just replicate.
  Mutate and Evolve.
  - Everyone *knew* visual programming languages had failed — until they didn’t.
  - What we have today has been built for the few.

We need to build more, try more, and involve *everyone*

Some of the Collaborators on This Work

- Barbara Ericson, Miranda Parker, Kathryn Cunningham, Amber Solomon, Richard Catrambone, Lauren Margulieux, Betsy DiSalvo, Tom McKlin, Rick Adrion, Renee Fall, Sarah Dunton, Brad Miller, Ria Galanos, Brian Dorn, and Briana Morrison

- Our Funders:
  US National Science Foundation

- http://computinged.wordpress.com
- http://guzdial.engin.umich.edu

Special Interest Group on Computer Science Education

Our Mission: To provide a global forum for educators to discuss research and practice related to the learning and teaching of computing…at all education levels

Thank you!

SPARE SLIDES

Definition of Computer Science

Computer Science

Professors of computer science are often asked: "Is there such a thing as computer science, and if there is, what is it?" The questions have a simple answer:

Wherever there are phenomena, there can be a science to describe and explain those phenomena. Thus, the simplest (and correct) answer to "What is botany?" is, "Botany is the study of plants." And zoology is the study of animals, astronomy the study of stars, and so on. Phenomena breed sciences.

There are computers. Ergo, computer science is the study of computers. The phenomena surrounding computers are varied, complex, rich. It remains only to answer the objections posed by many skeptics.

Objection 1. Only natural phenomena breed sciences, but computers are artificial, hence are whatever they are made to be, hence obey no invariable laws, hence cannot be described and explained. *Answer*. 1. The objection is patently false, since computers and computer programs are being described and explained daily. 2. The objection would equally rule out of science large portions of organic chemistry (substitute "silicones" for "computers"), physics (substitute "superconductivity" for "computers"), and even zoology (substitute "hybrid corn" for "computers"). The objection would certainly rule out mathematics, but in any event its status as a natural science is idiosyncratic.

Objection 2. The term "computer" is not well defined, and its meaning will change with new developments, hence computer science does not have a well-defined subject matter. *Answer*. The phenomena of all sciences change over time; the process of understanding assures that this will be the case. Astronomy did not originally include the study of interstellar gases; physics did not include radioactivity; psychology did not include the study of animal behavior. Mathematics was once defined as the "science of quantity."

Objection 3. Computer science is the study of algorithms (or programs), not computers. *Answer*. 1. Showing deeper insight than they are sometimes credited with, the founders of the chief professional organization for computer science named it the Association for Computing Machinery. 2. In the definition, "computers" means "living computers"—the hardware, their programs or algorithms, and all that goes with them. Computer science is the study of the phenomena surrounding computers. "Computers plus algorithms," "living computers," or simply "computers" all come to the same thing—the same phenomena.

Objection 4. Computers, like thermometers, are instruments, not phenomena. Instruments lead away to their user sciences; the behaviors of instruments are subsumed as special topics in other sciences (not always the user sciences—electron microscopy belongs to physics, not biology). *Answer*. The computer is such a novel and complex instrument that its behavior is subsumed under no other science; its study does not lead away to user sciences, but to further study of computers. Hence, the computer is not just an instrument but a phenomenon as well, requiring description and explanation.

Objection 5. Computer science is a branch of electronics (or mathematics, psychology, and so forth). *Answer*. To study computers, one may need to study some or all of these. Phenomena define the focus of a science, not its boundaries. Many of the phenomena of computers are also phenomena of some other science. The existence of biochemistry denies neither the existence of biology nor of chemistry. But all of the phenomena of computers are not subsumed under any one existing science.

Objection 6. Computers belong to engineering, not science. *Answer*. They belong to both, like electricity (physics and electrical engineering) or plants (botany and agriculture). Time will tell what professional specialization is desirable between analysis and synthesis, and between the pure study of computers and their application.

Computer scientists will often join hands with colleagues from other disciplines in common endeavor. Mostly, computer scientists will study living computers with the same passion that others have studied plants, stars, glaciers, dyestuffs, and magnetism; and with the same confidence that intelligent, persistent curiosity will yield interesting and perhaps useful knowledge.

ALLEN NEWELL
ALAN J. PERLIS
HERBERT A. SIMON
Graduate School of Industrial Administration, Carnegie Institute of Technology, Pittsburgh, Pennsylvania 15213

"The Big Trouble with Scientific Writing . . ."

When I see articles, as I frequently do these days, exhorting authors to greater simplicity and clarity (*1*), I think of the first little scientific note I wrote, when I was an idealistic graduate student. I wrote it as simply and directly as I could. It began, "The big trouble with diffusion cloud chambers is low radiation resistance," and it went on in the same vein. My co-workers thought it needed a little more work. Secretly I did not agree, so I decided to attempt to make it into a parody of

Science, 22 Sept 1967

First use of the term "Computer Science"

In 1961 we find him using the term "computer science" for the first time in his writing:

> [Computers] are developing so rapidly that even computer scientists cannot keep up with them. It must be bewildering to most mathematicians and engineers...In spite of the diversity of the applications, the methods of attacking the difficult problems with computers show a great unity, and the name of Computer Sciences is being attached to the discipline as it emerges. It must be understood, however, that this is still a young field whose structure is still nebulous. The student will find a great many more problems than answers. [59, p. 177]

He identified the "computer sciences" as the theory of programming, numerical analysis, data processing, and the design of computer systems, and observed that the latter three were better understood than the theory of programming, and more available in courses.

Knuth, *CACM* 1972

Forsythe on “Computer Science” (1961)

During that academic year he lectured on “Educational Implications of the Computer Revolution” at Brown University:

“Machine-held strings of binary digits can simulate a great many *kinds* of things, of which numbers are just one kind. For example, they can simulate automobiles on a freeway, chess pieces, electrons in a box, musical notes, Russian words, patterns on a paper, human cells, colors, electrical circuits, and so on. To think of a computer as made up essentially of numbers is simply a carry-over from the successful use of mathematical analysis in studying models...Enough is known already of the diverse applications of computing for us to recognize the birth of a coherent body of technique, which I call *computer science*...Whether computers are used for engineering design, medical data processing, composing music, or other purposes, the structure of computing is much the same. We are extremely short of talented people in this field, and so we need departments, curricula, and research and degree programs in computer science...I think of the Computer Science Department as eventually including experts in Programming, Numerical Analysis, Automata Theory, Data Processing, Business Games, Adaptive Systems, Information Theory, Information Retrieval, Recursive Function Theory, Computer Linguistics, etc., as these fields emerge in structure...Universities must respond [to the computer revolution] with far-reaching changes in the educational structure. [60]