

# Learning to code

why we fail, how we flourish

Andrew J. Ko, Ph.D.  
Code & Cognition Lab  
The Information School



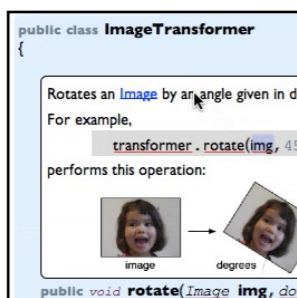
# Me

- Professor for the last ~10 years at **UW Seattle**
- Ph.D. from **Carnegie Mellon's** HCI Institute
- Background in **CS, Psychology,** and **Design**



# Code is the most powerful, least usable interface we've invented

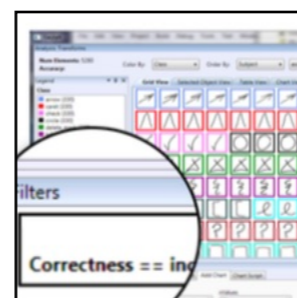
- Everyone that *wants* to code should be able to
- But there are immense learning barriers
- I spent the first decade trying to lower these barriers by creating more *usable interactive developer tools*



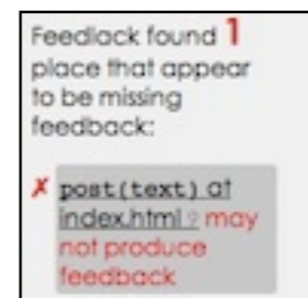
read



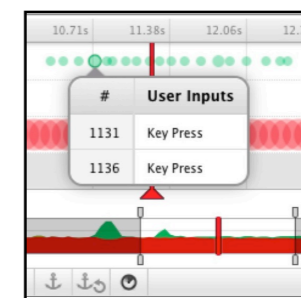
write



test



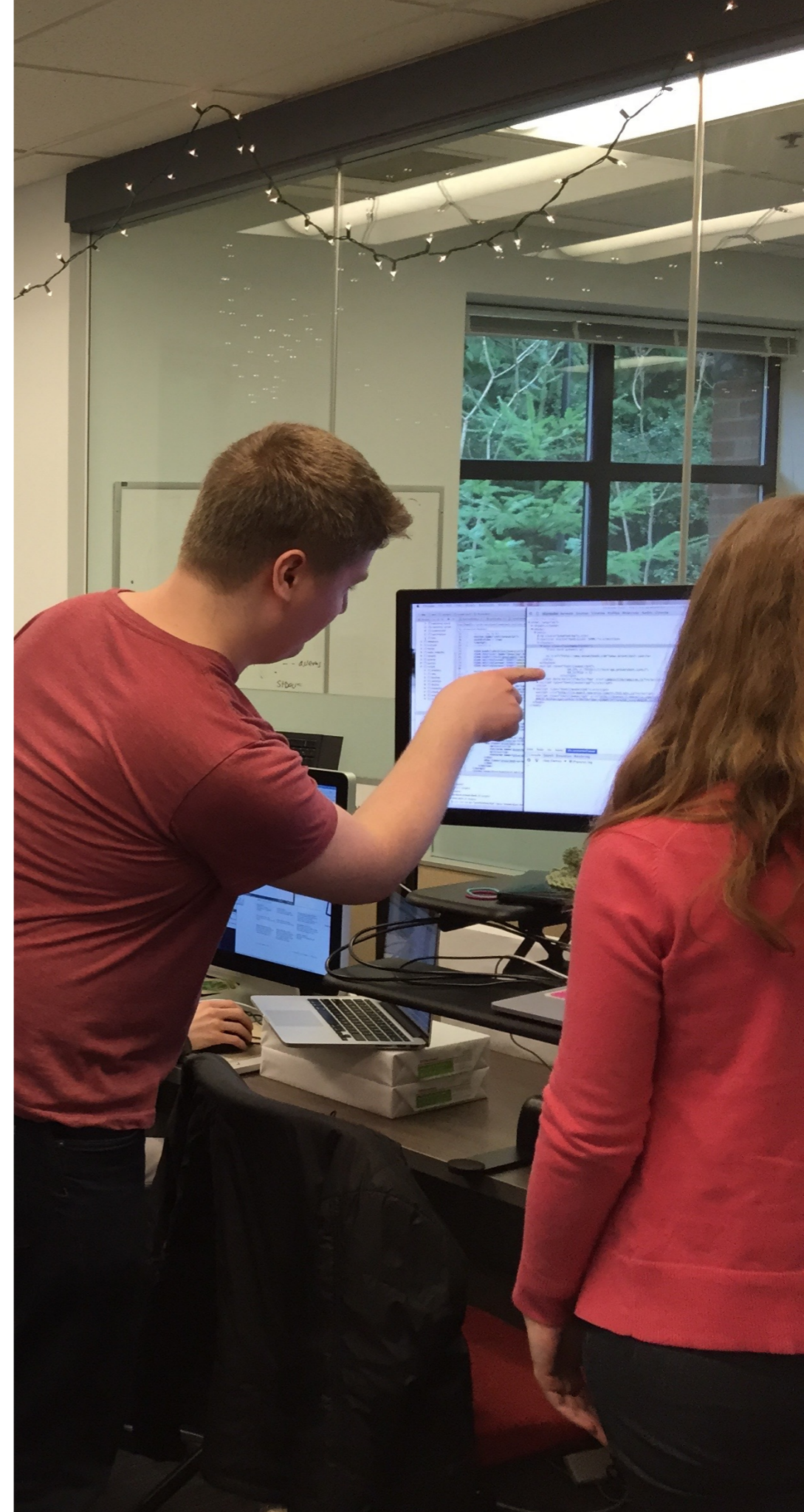
verify



debug

# Skills > tools

- I spent 3 years as CTO managing ~8 developers at AnswerDash
- What I saw:
  - Tools only *amplify* skills
  - Skills come from learning
  - Learning comes from teaching
  - I spent most of my time teaching



Millions want to learn to code

# Millions want to learn to code



Ada | DEVELOPERS ACADEMY

MEMBER  
CsforALL  
CONSORTIUM



amazon

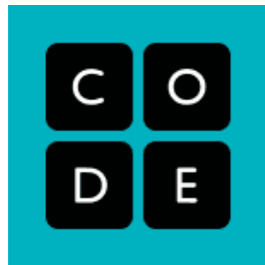
Microsoft



csnyc

coursera

facebook



COMPUTING AT SCHOOL  
EDUCATE · ENGAGE · ENCOURAGE  
Part of BCS, The Chartered Institute for IT

Google

SCRATCH

KHANACADEMY

accenture

salesforce

CSEd  
WEEK

\$1.3 billion

# Are people learning?

we have (some) evidence

# 77% of Code.org's 500 million K-12 learners complete 0-2 puzzles code.org



Minecraft: Hero's Journey

1

I've finished my Hour of Code

Sign in



## MINECRAFT



The door is locked, but the Agent is here to help!

Snap a `move forward` block to the bottom of the `when run` block in the workspace to get the Agent to the pressure plate, then press "Run" and use the arrow keys to move out of the house to collect the chest.

Less

Blocks

Workspace:

Start Over

Show Code

move forward

when run

Run



Need help? See these videos and hints



Record enrollment in AP CS, but most don't take exam, and 60% who do, fail it (especially underrepresented minorities) College Board



After a year of intro courses, most undergrads can't accurately predict the outcome of simple programs, or solve simple programming problems McCracken et al. 2001, Lister et al. 2004 et al. 2013, Seppälä et al. 2015



In 2017, 23,000 adults in 95 U.S. coding bootcamps; 24 report dropout rates of 10-50% [CourseReport.com](http://CourseReport.com)



62% of employers view applicants to entry-level developer positions as lacking basic programming knowledge Career Advisory Board Survey 2016



# So I did some reading.

- Read seminal literature in learning science and education e.g. How people learn: Brain, mind, experience, and school
- Read 30 years of computing education research:
  - *ACM International Computing Education Research Conference (ICER)*
  - *ACM Transactions on Computing Education (TOCE)*
  - *SIGCSE Technical Symposium (SIGCSE)*

# Why people fail to learn to code

1. People find computing boring, solitary, unwelcoming
2. People struggle to learn their first programming language
3. People struggle to solve programming problems
4. Teachers struggle to teach these things
5. Teachers blame learners for failure
6. People lose confidence and quit

# My goal

1. People find computing boring, solitary, unwelcoming
2. People struggle to learn their first programming language
3. People struggle to solve programming problems
  - Why are these hard?
  - What are effective, equitable, and scalable ways for people to learn these skills?

# This talk

- Why learning to program is hard (3 studies)
- Making programs easier to read (1 theory, 3 ideas)
- Making programs easier to write (1 theory, 1 idea)



Why is learning to  
program difficult?



# Study 1 – 70 high school teens



# High school

Ko, A.J. and Davis, K. (2017). Computing Mentorship in a Software Boomtown: Relationships to Adolescent Interest and Beliefs. ACM ICER.

Ko, A.J. et al. (2018). Informal Mentoring of Adolescents about Computing: Relationships, Roles, Qualities, and Impact. ACM SIGCSE.

- Many teens lacked feedback or support about their learning from teachers and family:
  - *He do not spent much time with me to be able to understand my problem in the class or unable to help me on it... throughout the AP class I would cried myself to sleep in silent without letting my older brother know my struggle... (M, Asian, 17)*

# High school

Ko, A.J. and Davis, K. (2017). Computing Mentorship in a Software Boomtown: Relationships to Adolescent Interest and Beliefs. ACM ICER.

Ko, A.J. et al. (2018). Informal Mentoring of Adolescents about Computing: Relationships, Roles, Qualities, and Impact. ACM SIGCSE.

- Some teens had *informal computing mentors* who provided encouraging instruction and feedback.
  - Associated with stronger interest in learning to code, independent of gender, socioeconomic status.
- Teens sought teachers and mentors who:
  - Would not judge them for their failures
  - Would inspire them to learn
  - Had the expertise to guide them



# Study 2 – 26 Bootcamps attendees



# Bootcamps

Thayer, K. and Ko, A.J. (2017). Barriers Faced by Coding Bootcamp Students. ACM ICER.

- Some bootcamps were inclusive and encouraging, but many offered no instruction or feedback:
  - *So they're trying to get you into this mentality of you have to read all the documentation. They sit back in the background [to let students read the documentation], and what annoys me is that I've paid a lot of money so that I could have somebody there to teach it to me.*

# Bootcamps

Thayer, K. and Ko, A.J. (2017). Barriers Faced by Coding Bootcamp Students. ACM ICER.

- Many bootcamps offered an unwelcoming culture for learners without prior knowledge:
  - *It was divided, the class. Those with experience, I think, they were looking down at [those of us without experience] because maybe there were certain things we were supposed to know and we didn't.*



# Study 3 – 30 Coding Tutorials

Learn to code interactively, for free.



Sign Up

Log In

Choose a username

Your email address

Choose a password

I'm not a robot



reCAPTCHA



# Tutorials

Kim, A. and Ko, A.J. (2017). A Pedagogical Analysis of Online Coding Tutorials. ACM SIGCSE.

- Four learning science principles
  1. Connect instruction to prior knowledge
  2. Organize declarative knowledge
  3. Offer personalized feedback on practice
  4. Foster self-regulation in problem solving
- Ada completed all 30 tutorials across 100+ hours, judging every lesson against these principles

# Tutorials

Kim, A. and Ko, A.J. (2017). A Pedagogical Analysis of Online Coding Tutorials. ACM SIGCSE.

- Most tutorials failed to meet all them:
  - ✘ 1. No connection to prior knowledge
  - ✘ 2. No organization of declarative knowledge about programming languages
  - ✘ 3. No personalized feedback on program correctness or errors
  - ✘ 4. No instruction on how to solve programming problems.

# Why is learning to program difficult?

Few of these contexts actually *teach* programming. There are many opportunities to *read and write code*, but learners receive little feedback on whether they are reading or writing *correctly*.

Making programs  
easier to *read*

*One theory, three ideas*

# Extant theories about why understanding programs is hard

- Wrong programming language
  - Static typing, syntax, and errors matter, but only a little (e.g., Stefik & Siebert 2013)
- Wrong IDE
  - Relative to text, drag and drop “blocks” editors reduce dropout, but don’t improve learning (Cooper et al. 2001)
- Wrong biology
  - No evidence of “geek gene” or bimodal grade distributions (Patitsas et al. 2016)

# Extant theories about why understanding programs is hard

- Wrong programming language
  - Static typing, no max, and errors that are not caught by a compiler (e.g., Stefik & Weber, 2013)
- Wrong IDE
  - Relative to text, drag and drop “blocks” editors reduce dropouts, but do not improve learning (Copert et al., 2001)
- Wrong biology
  - No evidence of “geek gene” or bimodal grade distributions (Patitsas et al. 2016)

# A new definition of PL knowledge

Knowing a PL means:

1. Being able to reliably and accurately *predict* an arbitrary program's **operational semantics** without the aid of a runtime environment. (Reading a program and knowing what it will do).
2. Knowing **how syntax maps** onto operational semantics.

*Note that I'm excluding knowledge of common design patterns, architectures, tools, norms, etc. This strictly concerns the ability to accurately **read** programs.*

# An example

Knowing a JavaScript *if-statement* means knowing:

- 1** Condition is evaluated
  - 2** If it's true, all of the statements between the first set of braces are executed, and everything between the else braces are skipped.
  - 3** Otherwise, the statements in the first set are skipped, and the statements in the second set are executed.
- ```
if(dataIsValid && serviceIsOnline) {  
    submit();  
}  
else {  
    alert("Bad error message!");  
}
```
-



# Knowing a entire PL

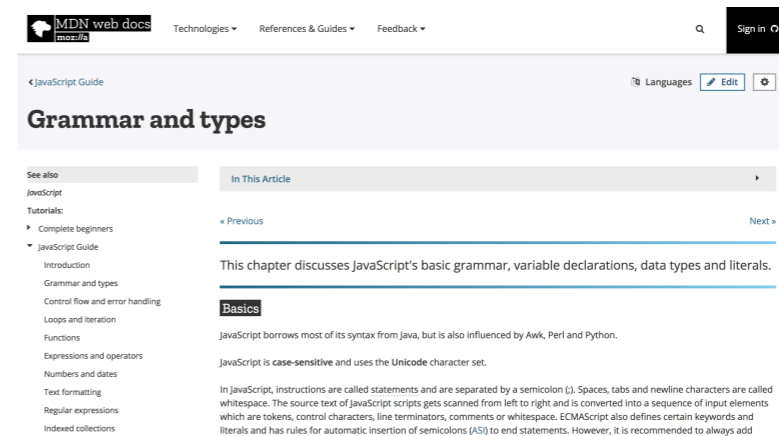
- Knowing *all* of JavaScript means knowing *all* of the semantics for JavaScript's entire grammar
- That's about **90 non-terminals** in the grammar, each with its own semantic nuances
- Most introductory programming courses never explain any of this:
  - In UW's CS1 course, the 1st homework is to write a Java program with function declarations, function calls, string literals. None of the lectures explain any of this, and, not surprisingly, most students fail.

# Four major pedagogies

Learn formal semantics

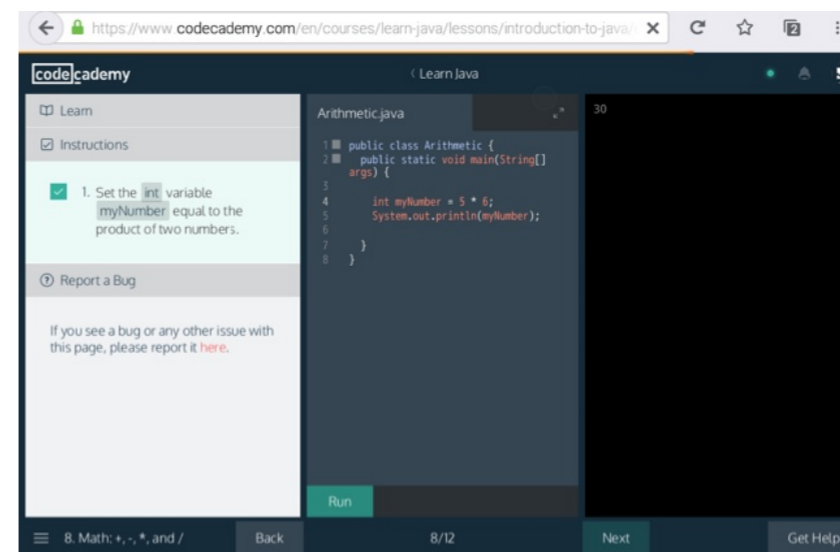
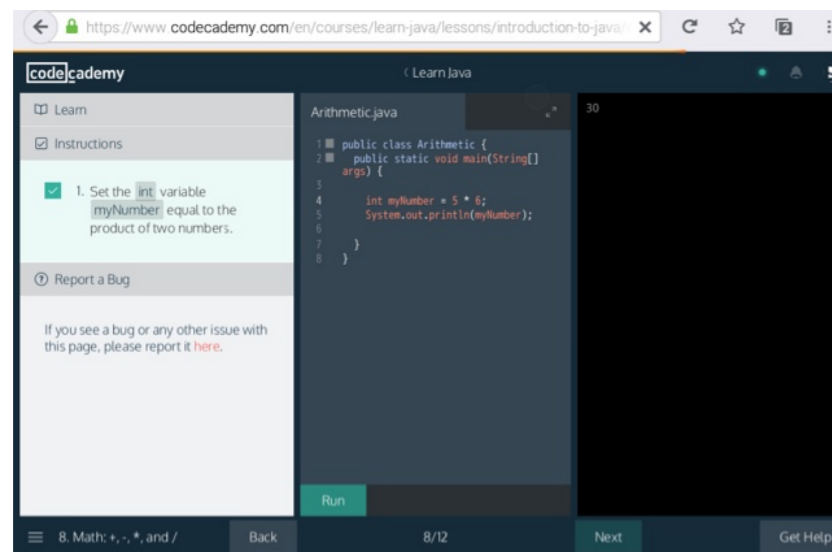
Explain via natural language

$$\frac{}{\mathbf{R} \vdash e_{empty} \Rightarrow \mathbf{R}} \text{ (EMPTY)} \quad \frac{e_2 \in \mathbf{R}}{\mathbf{R} \vdash e_1 = e_2 \Rightarrow \mathbf{R} \cup \{e_1\}} \text{ (ASSIGN)}$$
$$\frac{e_2 \in \mathbf{R}}{\mathbf{R} \vdash e_1.field^* = e_2 \Rightarrow \mathbf{R} \cup \{e_1\}} \text{ (ASSIGN-FIELD)} \quad \frac{e_2 \in \mathbf{R}}{\mathbf{R} \vdash e_1[*]^* = e_2 \Rightarrow \mathbf{R} \cup \{e_1\}} \text{ (ASSIGN-ARRAY)}$$
$$\frac{v \in \mathbf{R}; \text{pointer\_type\_p}(v); \mathbf{R} \vdash e \Rightarrow \mathbf{R}'}{\mathbf{R} \vdash v \oplus e \Rightarrow \mathbf{R}' \cup \{v \oplus e\}} \text{ (BINOP1)}$$
$$\frac{v \in \mathbf{R}'; \text{pointer\_type\_p}(v); \mathbf{R} \vdash e \Rightarrow \mathbf{R}'}{\mathbf{R} \vdash e \oplus v \Rightarrow \mathbf{R}' \cup \{e \oplus v\}} \text{ (BINOP2)} \quad \frac{\mathbf{R} \vdash e_1 \Rightarrow \mathbf{R}'; \mathbf{R}' \vdash e_2 \Rightarrow \mathbf{R}''}{\mathbf{R} \vdash e_1; e_2 \Rightarrow \mathbf{R}''} \text{ (SEQ)}$$
$$\frac{\mathbf{R} \vdash e_1 \Rightarrow \mathbf{R}_1; \mathbf{R}_1 \vdash e_2 \Rightarrow \mathbf{R}_2; \mathbf{R}_1 \vdash e_3 \Rightarrow \mathbf{R}_3; \mathbf{R}_2 \cup \mathbf{R}_3 \vdash e_4 \Rightarrow \mathbf{R}'}{\mathbf{R} \vdash \text{if}(e_1) \text{ then } \{e_2\} \text{ else } \{e_3\} e_4 \Rightarrow \mathbf{R}'} \text{ (IF)}$$
$$\frac{\text{function\_type\_p}(e) \cup \{v \mid \text{global\_p}(v)\} \vdash e.body \Rightarrow \mathbf{R}}{\{\} \vdash e \Rightarrow \mathbf{R}} \text{ (FUNCTION)}$$



Write code

Step through execution



# Four major pedagogies

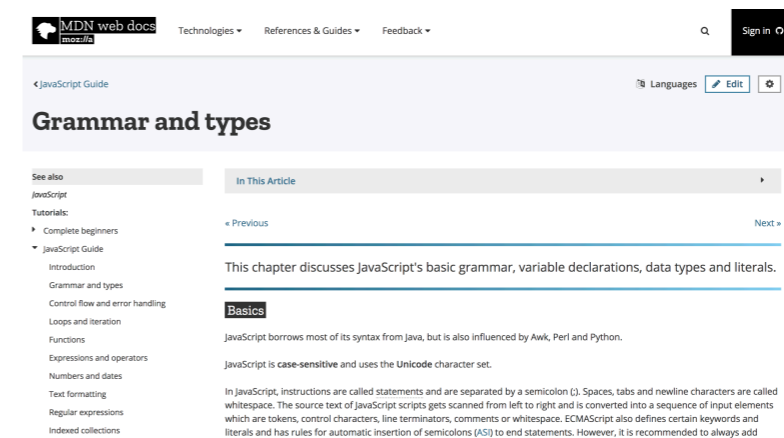
Learn formal semantics

Explain via natural language

$$\frac{}{\mathbf{R} \vdash e_{empty} \Rightarrow \mathbf{R}} \text{ (EMPTY)} \quad \frac{e_2 \in \mathbf{R}}{\mathbf{R} \vdash e_1 = e_2 \Rightarrow \mathbf{R} \cup \{e_1\}} \text{ (ASSIGN)}$$
$$\frac{e_2 \in \mathbf{R}}{\mathbf{R} \vdash e_{empty} \Rightarrow \mathbf{R}} \text{ (ASSIGN-FIELD)} \quad \frac{e_2 \in \mathbf{R}}{\mathbf{R} \vdash e_{empty} \Rightarrow \mathbf{R}} \text{ (ASSIGN-ARRAY)}$$

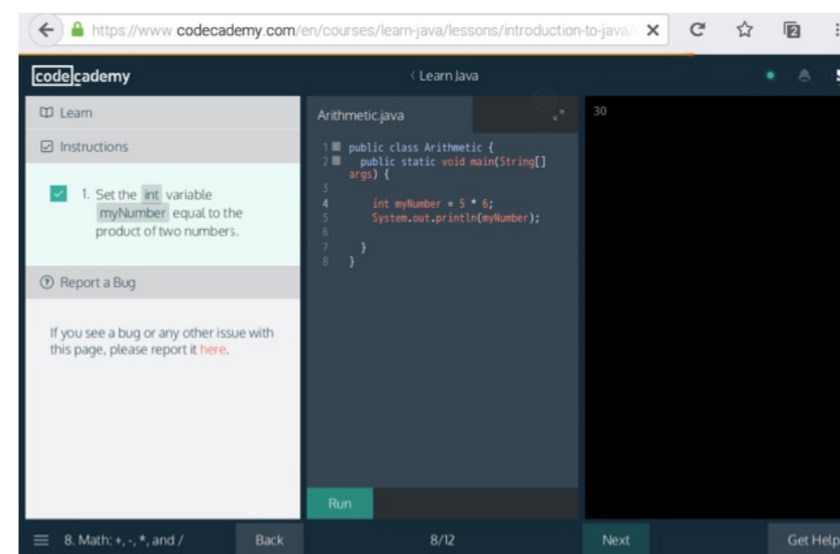
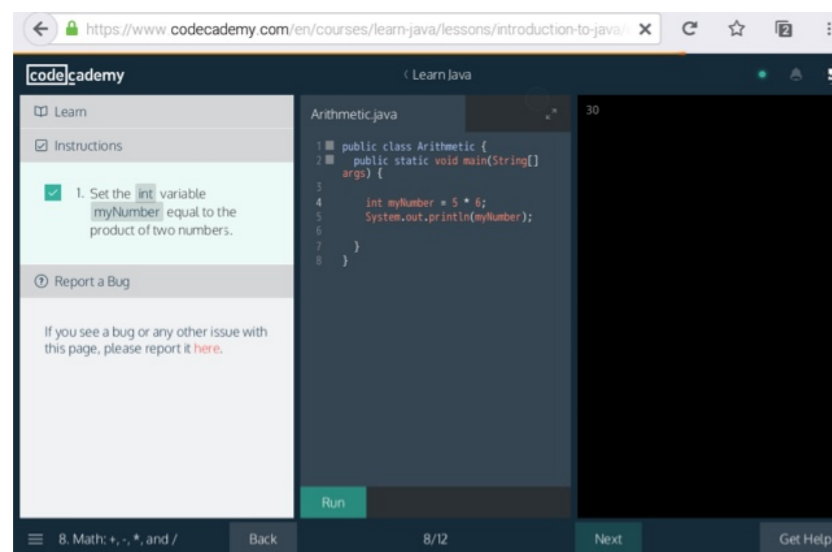
**No syntax mapping;  
Requires learning a notation**

$$\frac{\{e.args[i] \mid 0 < i < e.numargs \wedge \text{pointer\_type.p}(e.args[i])\} \cup \{v \mid \text{global.p}(v)\} \vdash e.body \Rightarrow \mathbf{R}}{\{\} \vdash e \Rightarrow \mathbf{R}} \text{ (FUNCTION)}$$



Write code

Step through execution



# Four major pedagogies

Learn formal semantics

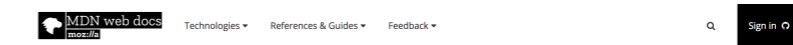
Explain via natural language

$$\frac{}{\mathbf{R} \vdash e_{empty} \Rightarrow \mathbf{R}} \text{ (EMPTY)} \quad \frac{e_2 \in \mathbf{R}}{\mathbf{R} \vdash e_1 = e_2 \Rightarrow \mathbf{R} \cup \{e_1\}} \text{ (ASSIGN)}$$

$$\frac{e_2 \in \mathbf{R}}{\mathbf{R} \vdash e \Rightarrow \mathbf{R}} \text{ (ASSIGN-FIELD)} \quad \frac{e_2 \in \mathbf{R}}{\mathbf{R} \vdash e \Rightarrow \mathbf{R}} \text{ (ASSIGN-ARRAY)}$$

**No syntax mapping;  
Requires learning a notation**

$$\frac{\{e.args[i] \mid 0 < i < e.numargs \wedge \text{pointer\_type\_p}(e.args[i])\} \cup \{v \mid \text{global\_p}(v)\} \vdash e.body \Rightarrow \mathbf{R}}{\{\} \vdash e \Rightarrow \mathbf{R}} \text{ (FUNCTION)}$$



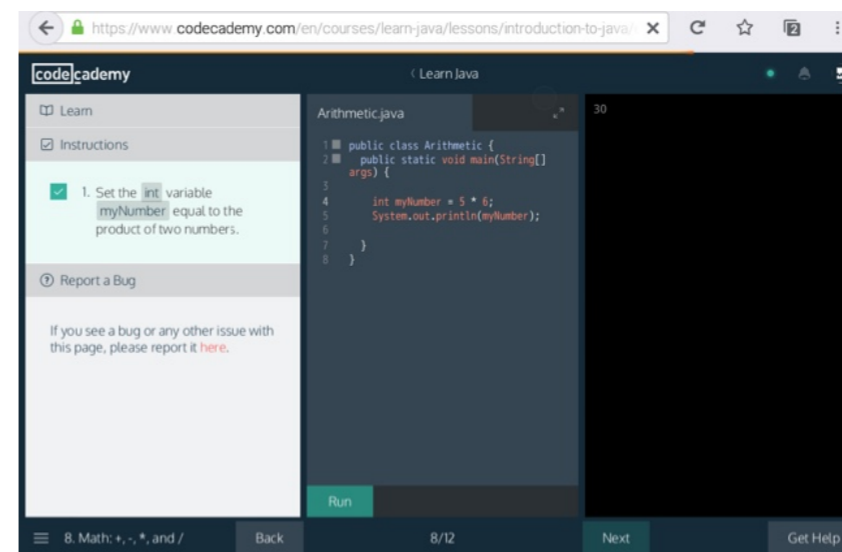
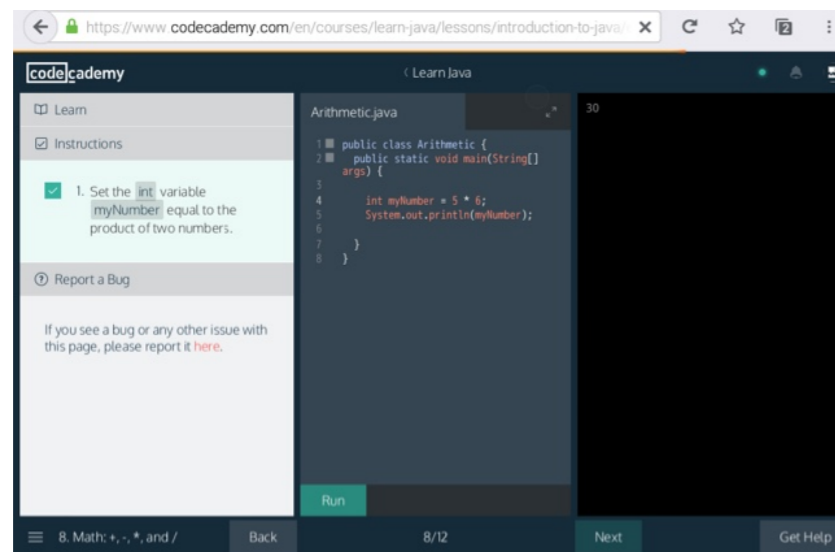
**Ambiguous, weak  
syntax mapping**

Numbers and dates  
Text formatting  
Regular expressions  
Indexed collections

In JavaScript, instructions are called statements and are separated by a semicolon (;). Spaces, tabs and newline characters are called whitespace. The source text of JavaScript scripts gets scanned from left to right and is converted into a sequence of input elements which are tokens, control characters, line terminators, comments or whitespace. ECMAScript also defines certain keywords and literals and has rules for automatic insertion of semicolons (ASI) to end statements. However, it is recommended to always add

Write code

Step through execution



# Four major pedagogies

Learn formal semantics

Explain via natural language

$$\frac{}{\mathbf{R} \vdash e_{empty} \Rightarrow \mathbf{R}} \text{ (EMPTY)} \quad \frac{e_2 \in \mathbf{R}}{\mathbf{R} \vdash e_1 = e_2 \Rightarrow \mathbf{R} \cup \{e_1\}} \text{ (ASSIGN)}$$

**No syntax mapping;  
Requires learning a notation**

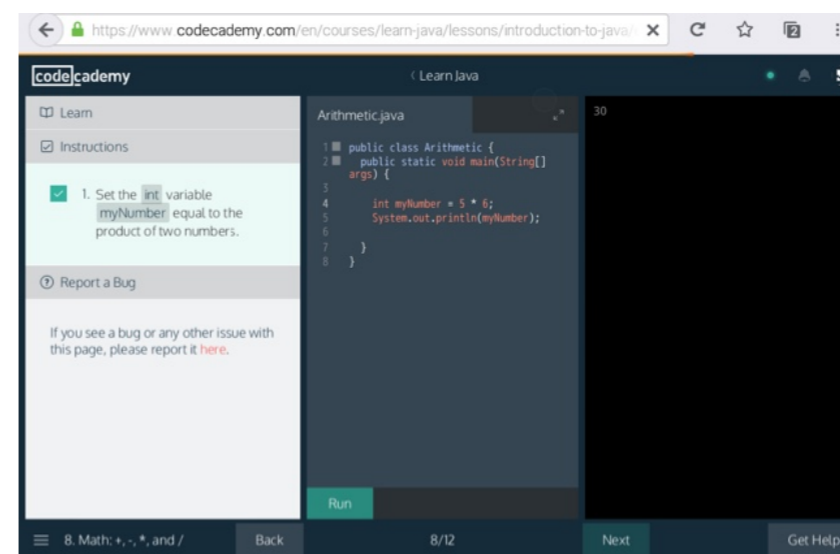
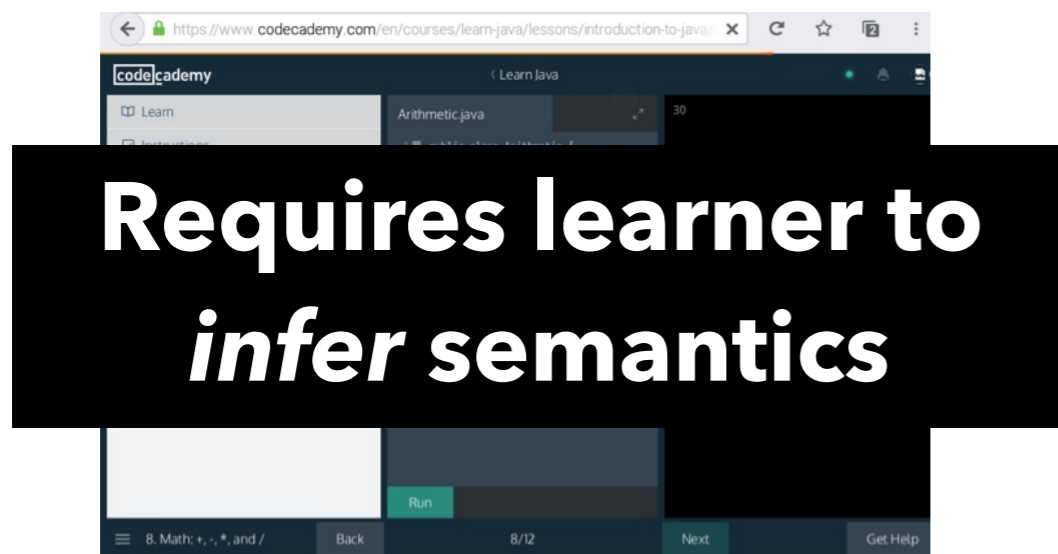
$$\frac{\frac{\{e.args[i] \mid 0 < i < e.numargs \wedge pointer\_type.p(e.args[i])\} \cup \{v \mid global.p(v)\} \vdash e.body \Rightarrow \mathbf{R}}{\{\} \vdash e \Rightarrow \mathbf{R}}}{e_2 \in \mathbf{R}} \text{ (ASSIGN-FIELD)} \quad \frac{}{\mathbf{R} \vdash e_1 = e_2 \Rightarrow \mathbf{R} \cup \{e_1\}} \text{ (ASSIGN-ARRAY)}$$

Write code



**Ambiguous, weak  
syntax mapping**

Step through execution



# Four major pedagogies

Learn formal semantics

Explain via natural language

$$\frac{}{\mathbf{R} \vdash e_{\text{empty}} \Rightarrow \mathbf{R}} \text{ (EMPTY)} \quad \frac{e_2 \in \mathbf{R}}{\mathbf{R} \vdash e_1 = e_2 \Rightarrow \mathbf{R} \cup \{e_1\}} \text{ (ASSIGN)}$$

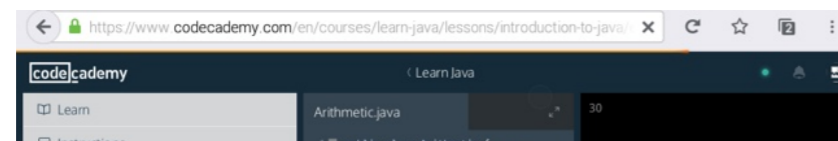
**No syntax mapping;  
Requires learning a notation**

$$\frac{\{e.\text{args}[i] \mid 0 < i < e.\text{numargs} \wedge \text{pointer\_type.p}(e.\text{args}[i])\} \cup \{v \mid \text{global.p}(v)\} \vdash e.\text{body} \Rightarrow \mathbf{R}}{\{\} \vdash e \Rightarrow \mathbf{R}} \text{ (FUNCTION)}$$

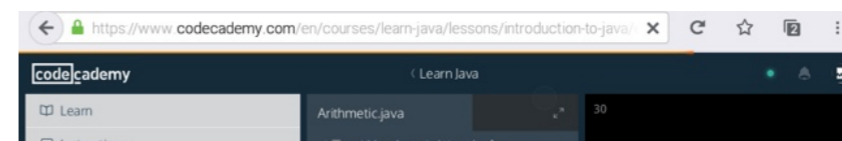
**Ambiguous, weak  
syntax mapping**

Write code

Step through execution



**Requires learner to  
*infer* semantics**



**Masks semantics within a  
line of code**

# How *should* we teach syntax semantics?

Teach a “notional machine” du Boulay 1989

1. Show *each step* of semantics and their **explicit effects** on the program counter, call stack, and memory
2. **Map** semantics to concrete syntax, creating an association between syntax and its side effects

# Three ideas

## Gidget



Mike Lee, Ph.D.

Learners *discover*  
semantics through  
debugging

## PLTutor



Greg Nelson

Tutor explicitly  
*teaches*  
semantics

## Tracing Strategies



Benji Xie

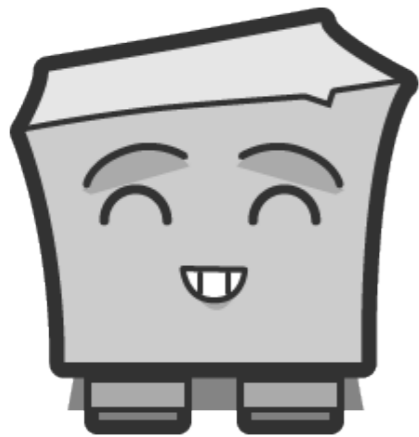
Learner  
reminded to  
follow semantics



# Gidget helpgidget.org



Mike Lee



- Frame coding as a **collaboration** between a person and computer
- Give learner a sequence of **debugging puzzles**
- Guide learners' attention to **contextualized instruction on syntax and semantics** of it's Pythonic language

# Gidget

helpgidget.org Lee et al. 2014, VL/HCC



Mike Lee

The screenshot shows the homepage of the Gidget website. At the top left, there are social media icons for Facebook and Twitter. In the top right corner, there are buttons for "Sign Up" and "Login". The main heading "Gidget" is centered in a large, bold, black font against a light blue sky background with white clouds. Below the heading is a green field containing various cartoon characters and objects: a grey robot-like character, an orange cat, a pink pig, a white mouse, a yellow chick, a red car, a yellow lion, a red apple, a brown basket, a green tree, and a blue ice cube. To the right of the field are three buttons: "PLAY!" (orange), "Level Editor" (dark brown with a yellow padlock icon), and "About" (orange). At the bottom left, there is a "donate" button. At the bottom center, there is a speaker icon and logos for NJIT, W, OSU, and NSF. At the bottom right, there is a "Q&A" button.

# Gidget

helpgidget.org Lee et al. 2014, VL/HCC



Mike Lee

The screenshot shows the homepage of the Gidget website. At the top, there are social media icons for Facebook and Twitter, and buttons for 'Sign Up' and 'Login'. The main title 'Gidget' is centered in a large, bold font against a blue sky with white clouds. Below the title, a collection of colorful, cartoonish characters and objects is displayed on a green field, including a grey robot-like character, an orange cat, a pink pig, a white mouse, a yellow chick, a red car, a basket, a tree, and an apple. To the right of the characters are three buttons: 'PLAY!' (orange), 'Level Editor' (dark brown with a lock icon), and 'About' (orange). At the bottom, there is a 'donate' button, a speaker icon, and logos for NJIT, W, OSU, and NSF. A 'Q&A' button is located in the bottom right corner.

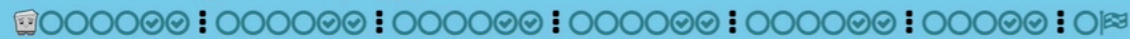
# Gidget

helpgidget.org Lee et al. 2014, VL/HCC

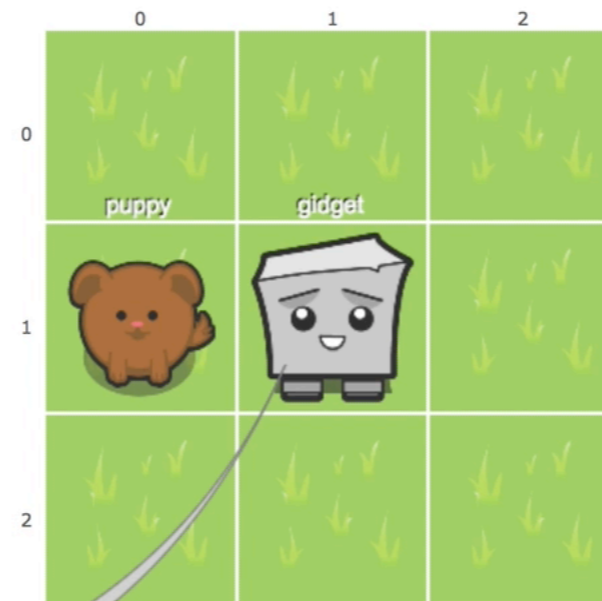


Mike Lee

Level 1. Let's get to the puppy!



## world



Make sure you always read the goals of the level on the bottom-left panel first, and then try running the code at least once using the buttons below the goals to see how the starting code works.

← Prev Next →

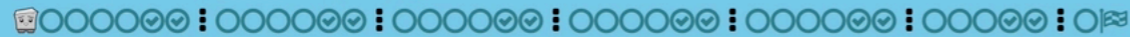
# Gidget

helpgidget.org Lee et al. 2014, VL/HCC



Mike Lee

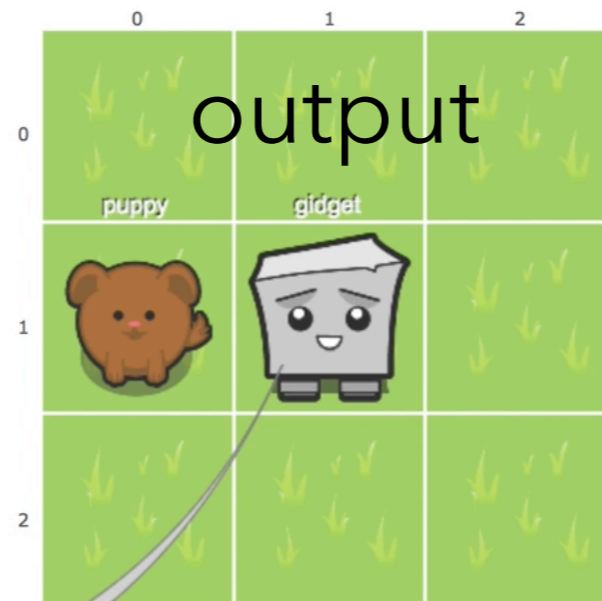
Level 1. Let's get to the puppy!



code  
editor

runtime  
state

world



Make sure you always read the goals of the level on the bottom-left panel first, and then try running the code at least once using the buttons below the goals to see how the starting code works.



test  
case

explanation

# Gidget [helpgidget.org](http://helpgidget.org) Lee et al. 2014, VL/HCC



Mike Lee

37 levels teaching 12 semantics, including formative assessments to verify understanding

**Level 1. Let's get to the puppy!**



**world**

|   | 0 | 1 |
|---|---|---|
| 0 |   |   |
|   |   |   |

# Gidget [helpgidget.org](http://helpgidget.org) Lee et al. 2014, VL/HCC



Mike Lee

37 levels teaching 12 semantics, including formative assessments to verify understanding

**Level 1. Let's get to the puppy!**



**world**



# Gidget [helpgidget.org](http://helpgidget.org) Lee et al. 2014, VL/HCC



Mike Lee

level 20 teaches function calls

Level 20. Press the button, open the gate!



## world

|   | 0      | 1 | 2      | 3 |
|---|--------|---|--------|---|
| 0 | basket |   | piglet |   |
| 1 |        |   |        |   |
| 2 |        |   |        |   |
| 3 |        |   |        |   |



# Gidget [helpgidget.org](http://helpgidget.org) Lee et al. 2014, VL/HCC



Mike Lee

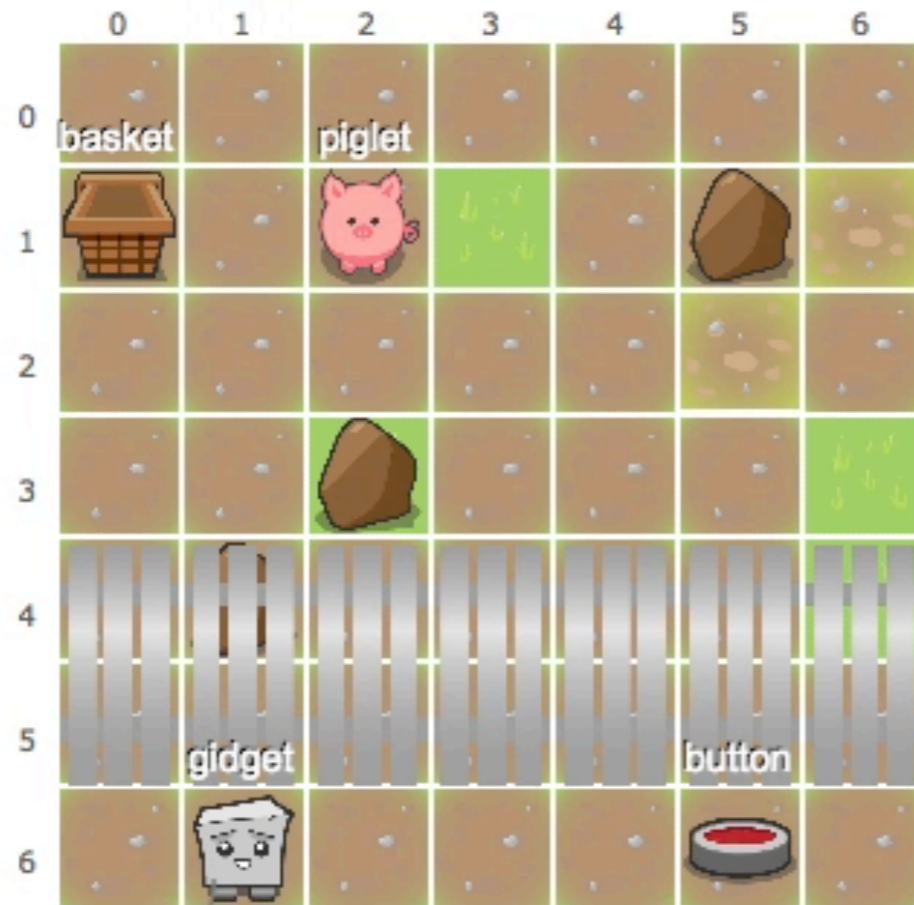
level 20 teaches function calls

Level 20. Press the button, open the gate!



## world

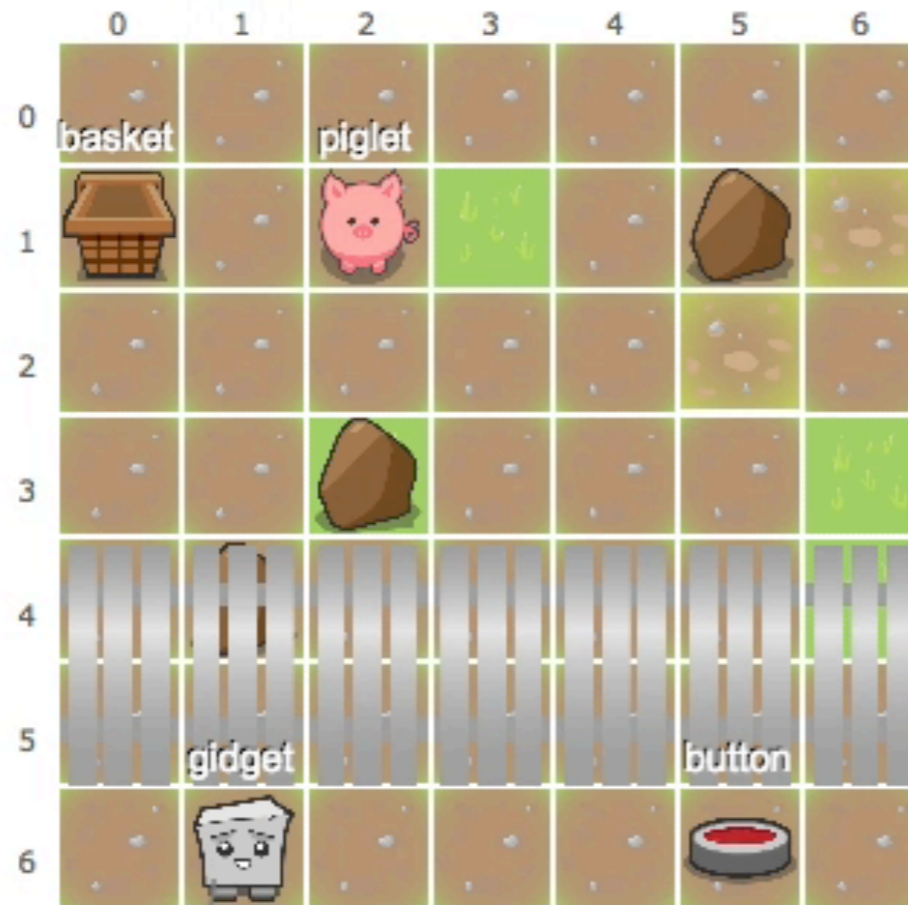
|   | 0      | 1 | 2      | 3 |
|---|--------|---|--------|---|
| 0 | basket |   | piglet |   |
| 1 |        |   |        |   |
| 2 |        |   |        |   |
| 3 |        |   |        |   |



All that **grabbing** & **dropping** made me remember a way to save time writing my programs though...

**functions!**

← Prev    Next →



All that **grabbing** & **dropping** made me remember a way to save time writing my programs though...

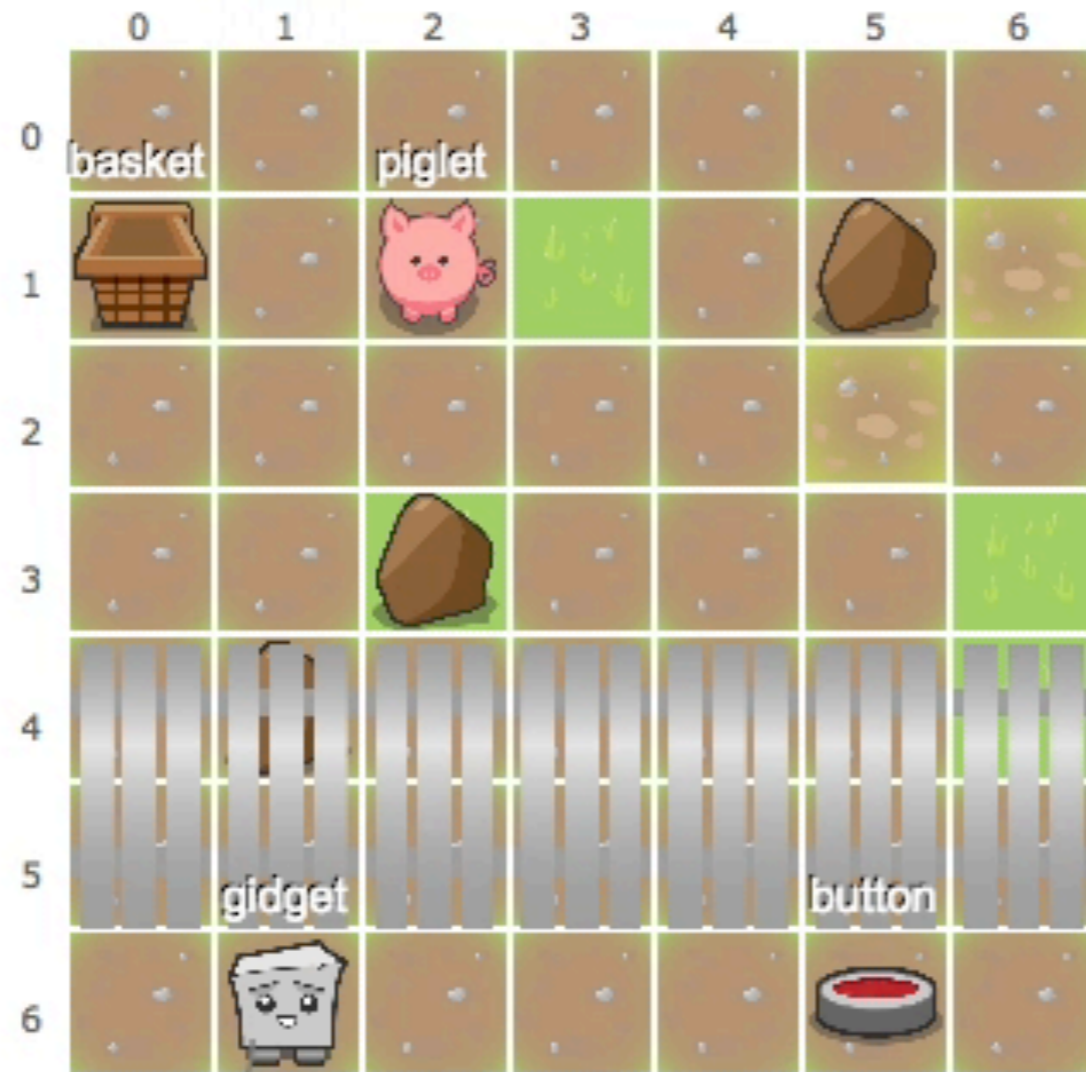
**functions!**

← Prev    Next →

Function  
definition  
semantics



# world



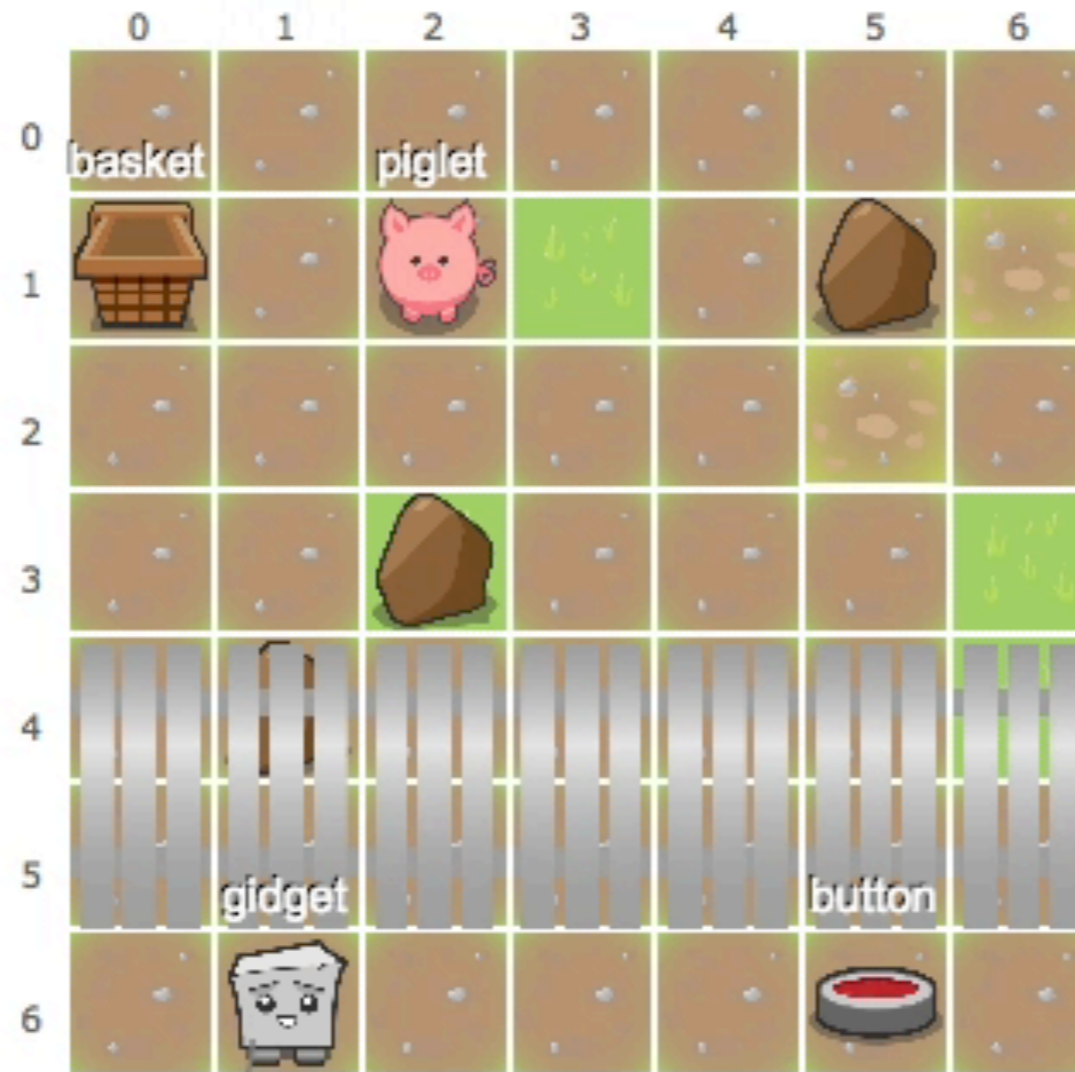
All that **grabbing** & **dropping** made me remember a way to save time writing my programs though... **functions!**

← Prev

Next →



# world



All that **grabbing** & **dropping** made me remember a way to save time writing my programs though... **functions!**

← Prev

Next →

Gidget explains syntax and semantics



# Gidget [helpgidget.org](http://helpgidget.org) Lee et al. 2014, VL/HCC



Mike Lee

Level 20. Press the button, open the gate!




The **green line**  
and Gidget's  
speech bubble  
maps syntax to  
semantics

## code

Original Code

Clear Code

```
goto /button/  
say "Let's click the button to see its functio  
/button/:openFence()  
function getPiglet()   
  goto /piglet/  
  set /piglet/:nickname to "wilbur"  
  set /piglet/:age to 3  
  grab /piglet/  
getBird()   
getThePiggy()   
goto /basket/
```

```
ensure /piglet/:nickname = "babe"  
ensure /piglet/:age = 3  
ensure # /piglet/ on /basket/ = 1
```

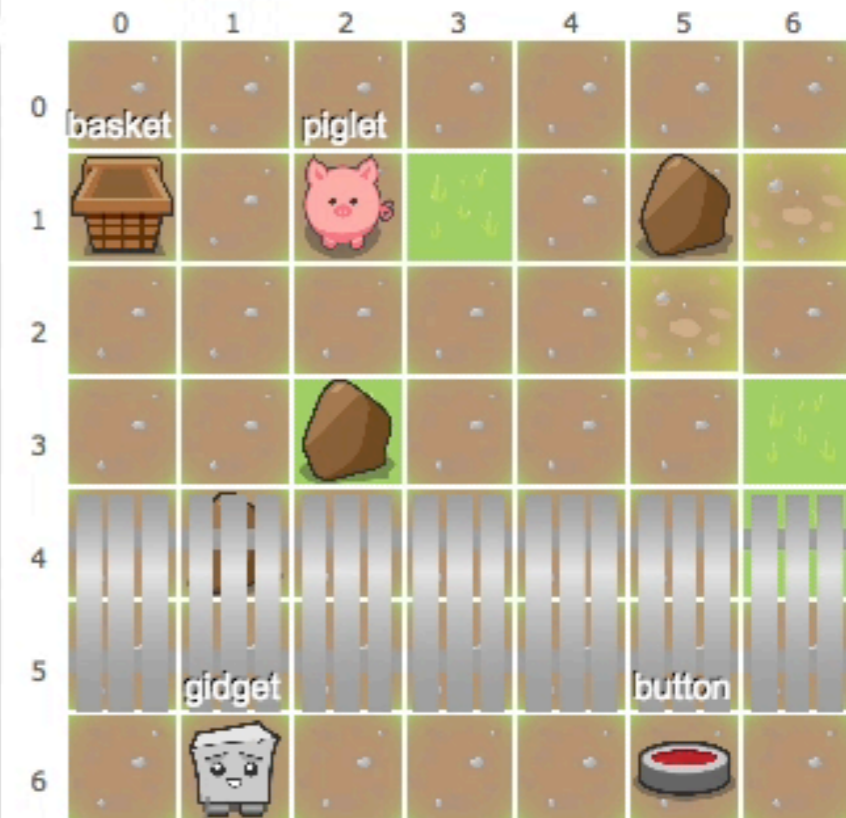
One step

One line

To end

Stop!

## world



Don't forget you can click on objects to see their properties, and you should try running my code first to see what happens!

 Prev      Next 

# Gidget

helpgidget.org Lee et al. 2014, VL/HCC



Mike Lee

Level 20. Press the button, open the gate!



The **green line**  
and Gidget's  
speech bubble  
maps syntax to  
semantics

## code

Original Code

Clear Code

```
goto /button/  
say "Let's click the button to see its functio  
/button/:openFence()  
function getPiglet()   
  goto /piglet/  
  set /piglet/:nickname to "wilbur"  
  set /piglet/:age to 3  
  grab /piglet/  
getBird()   
getThePiggy()   
goto /basket/
```

```
ensure /piglet/:nickname = "babe"  
ensure /piglet/:age = 3  
ensure # /piglet/ on /basket/ = 1
```

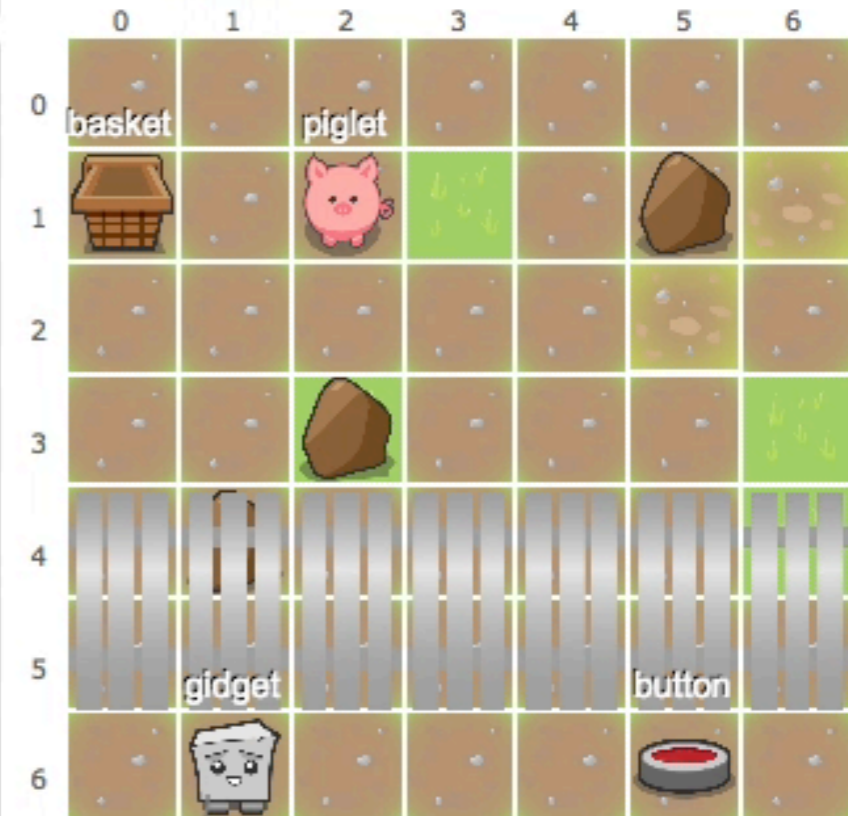
One step

One line

To end

Stop!

## world



Don't forget you can click on objects to see their properties, and you should try running my code first to see what happens!

 Prev      Next 

# Gidget

helpgidget.org Lee et al. 2014, VL/HCC



Mike Lee

Level 20. Press the button, open the gate!

## code

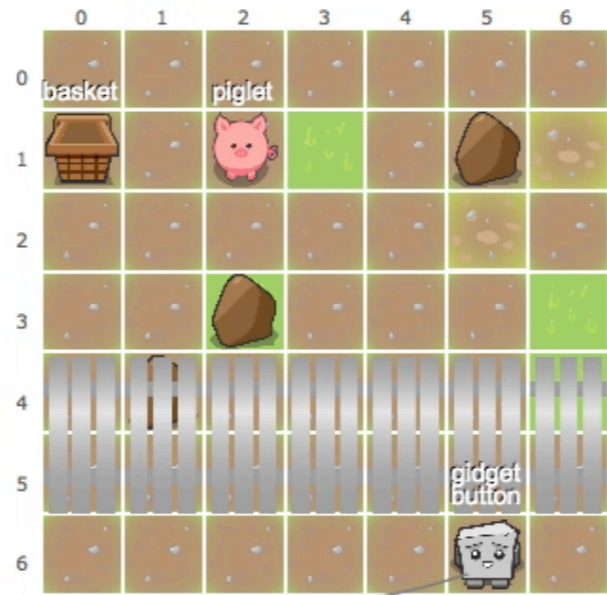
Original Code Clear Code

```
goto /button/  
say "Let's click the button to see its function"  
/button/:openFence()  
function getPiglet() ?  
  goto /piglet/  
  set /piglet/:nickname to "wilbur"  
  set /piglet/:age to 3  
  grab /piglet/  
getBird() ?  
getThePiggy() ?  
goto /basket/
```

```
ensure /piglet/:nickname = "babe"  
ensure /piglet/:age = 3  
ensure # /piglet/ on /basket/ = 1
```

One step One line To end Stop!

## world



Let's click the button to see its function name. It has to be exact!  
**Continue** →

## gidget



|              |           |
|--------------|-----------|
| energy       | 97        |
| grabbed      | [ ]       |
| image        | "default" |
| labeled      | true      |
| layer        | 1         |
| name         | "gidget"  |
| position     | [ 6, 5 ]  |
| rotation     | 0         |
| scale        | 1         |
| transparency | 1         |
| code()       | [ ]       |



# Gidget

helpgidget.org Lee et al. 2014, VL/HCC



Mike Lee

Level 20. Press the button, open the gate!

Progress indicator: 20/20 (all blue)

Icons: Book, Target, Menu

### code

Original Code Clear Code

```
goto /button/  
say "Let's click the button to see its function  
/button/:openFence()  
function getPiglet() ?  
  goto /piglet/  
  set /piglet/:nickname to "wilbur"  
  set /piglet/:age to 3  
  grab /piglet/  
getBird() ?  
getThePiggy() ?  
goto /basket/  
  
ensure /piglet/:nickname = "babe"  
ensure /piglet/:age = 3  
ensure # /piglet/ on /basket/ = 1
```

One step One line To end Stop!

### world

|   |        |   |        |   |   |   |   |  |
|---|--------|---|--------|---|---|---|---|--|
|   | 0      | 1 | 2      | 3 | 4 | 5 | 6 |  |
| 0 | basket |   | piglet |   |   |   |   |  |
| 1 |        |   |        |   |   |   |   |  |
| 2 |        |   |        |   |   |   |   |  |
| 3 |        |   |        |   |   |   |   |  |
| 4 |        |   |        |   |   |   |   |  |
| 5 |        |   |        |   |   |   |   |  |
| 6 |        |   |        |   |   |   |   |  |

Let's click the button to see its function name. It has to be exact!

**Continue** →

### gidget

|              |           |
|--------------|-----------|
| energy       | 97        |
| grabbed      | [ ]       |
| image        | "default" |
| labeled      | true      |
| layer        | 1         |
| name         | "gidget"  |
| position     | [ 6, 5 ]  |
| rotation     | 0         |
| scale        | 1         |
| transparency | 1         |
| code()       | [ ]       |

Gidget points out function definitions



```

to /button/
y "Let's click the button to see its function
button/:openFence()
function getPiglet() - ?
goto /piglet/
set /piglet/:nickname to "wilbur"
set /piglet/:age to 3
grab /piglet/
tBird() - ?
tThePiggy() - ?
to /basket/

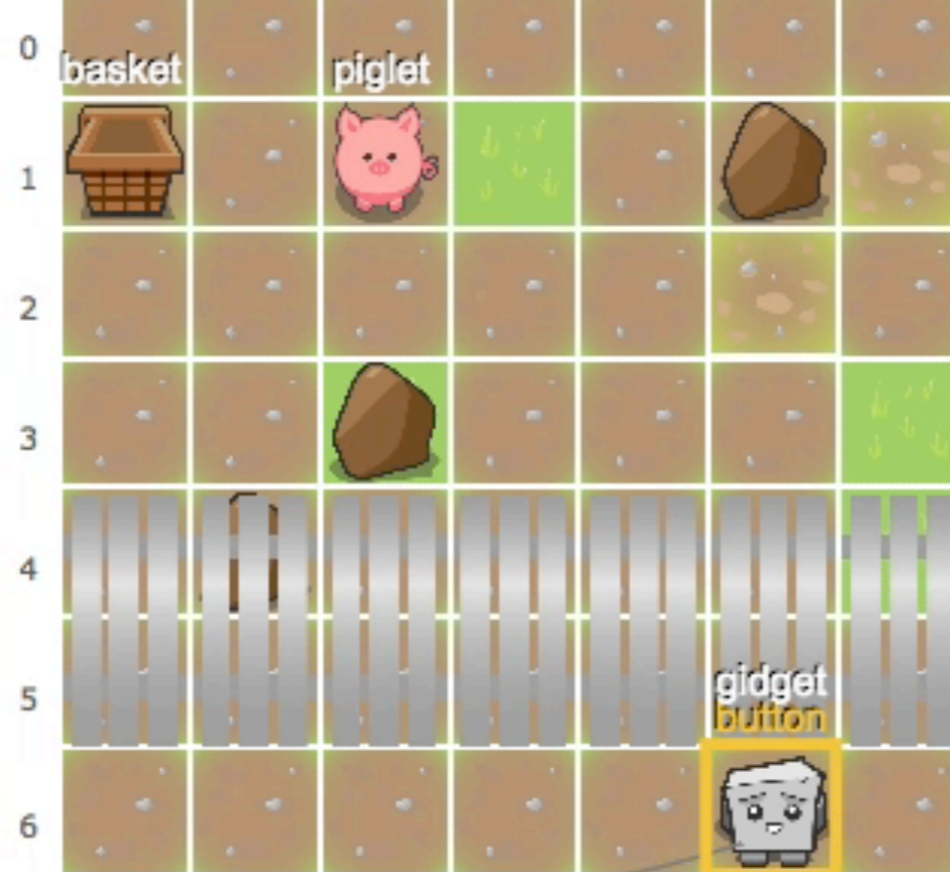
```

```

ensure /piglet/:nickname = "babe"
ensure /piglet/:age = 3
ensure # /piglet/ on /basket/ = 1

```

One step
One line
To end
Stop!



Let's click the button to see its function name. It has to be exact!

**Continue** →

```

energy          97
grabbed        [ ]
image          "default"
labeled        true
layer          1
name           "gidget"
position       [6, 5]
rotation       0
scale          1
transparency   1

code()
[ ]

```

## button

```

energy          100
grabbed        [ ]
image          "default"
labeled        true
layer          1
name           "button"
openGate       openGate()
position       [6, 5]
rotation       0
scale          1
transparency   1

code()

```

```

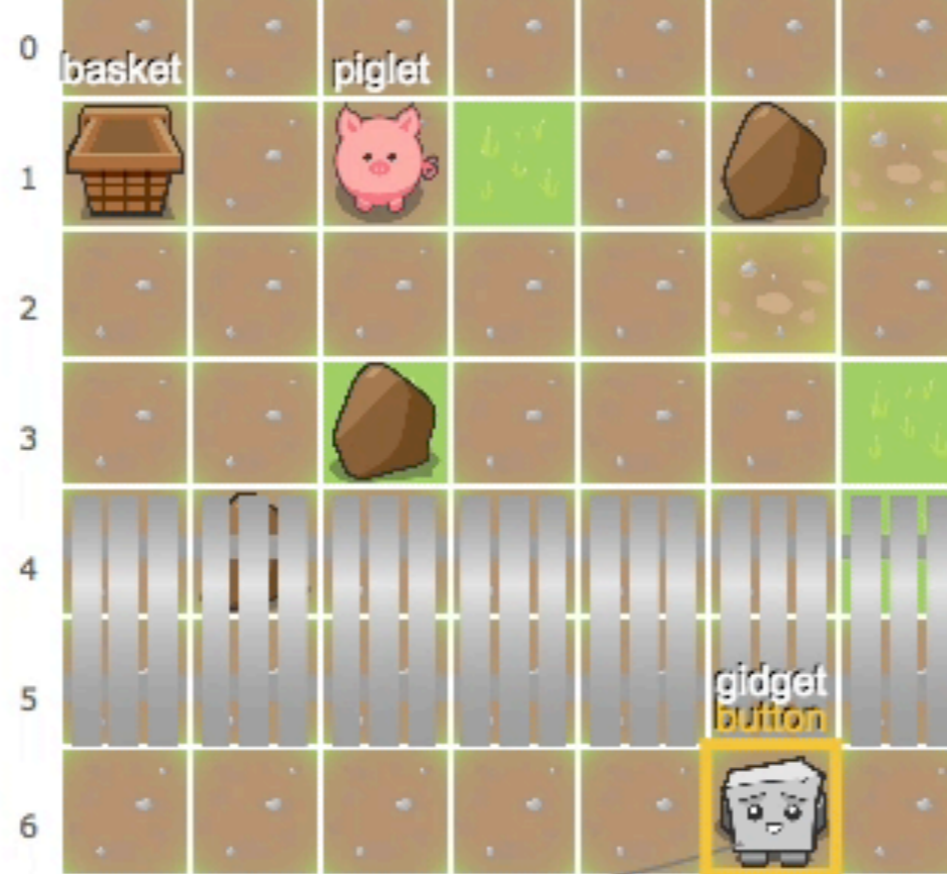
to /button/
y "Let's click the button to see its function
button/:openFence()
function getPiglet() -?
goto /piglet/
set /piglet/:nickname to "wilbur"
set /piglet/:age to 3
grab /piglet/
tBird() -?
tThePiggy() -?
to /basket/

```

```

ensure /piglet/:nickname = "babe"
ensure /piglet/:age = 3
ensure # /piglet/ on /basket/ = 1

```



Let's click the button to see its function name. It has to be exact!

**Continue** →

Gidget explains  
name resolution  
semantics

|               |           |
|---------------|-----------|
| energy        | 97        |
| grabbed       | [ ]       |
| image         | "default" |
| labeled       | true      |
| layer         | 1         |
| name          | "gidget"  |
| position      | [ 6, 5 ]  |
| rotation      | 0         |
| scale         | 1         |
| transparency  | 1         |
| <b>code()</b> | [ ]       |

## button

|               |                   |
|---------------|-------------------|
| energy        | 100               |
| grabbed       | [ ]               |
| image         | "default"         |
| labeled       | true              |
| layer         | 1                 |
| name          | "button"          |
| openGate      | <b>openGate()</b> |
| position      | [ 6, 5 ]          |
| rotation      | 0                 |
| scale         | 1                 |
| transparency  | 1                 |
| <b>code()</b> | [ ]               |

# Gidget [helpgidget.org](http://helpgidget.org) Lee et al. 2014, VL/HCC



Mike Lee

Level 20. Press the button, open the gate!

Progress indicator: 20/20 (all blue)

Icons: Home, Refresh, Menu

### code

Original Code Clear Code

```
goto /button/  
say "Let's click the button to see its function"  
/button/:openFence()  
function getPiglet() ?  
  goto /piglet/  
  set /piglet/:nickname to "wilbur"  
  set /piglet/:age to 3  
  grab /piglet/  
getBird() ?  
getThePiggy() ?  
goto /basket/  
  
ensure /piglet/:nickname = "babe"  
ensure /piglet/:age = 3  
ensure # /piglet/ on /basket/ = 1
```

One step One line To end Stop!

### world

|   |        |  |        |  |  |  |  |  |
|---|--------|--|--------|--|--|--|--|--|
| 0 | basket |  | piglet |  |  |  |  |  |
| 1 |        |  |        |  |  |  |  |  |
| 2 |        |  |        |  |  |  |  |  |
| 3 |        |  |        |  |  |  |  |  |
| 4 |        |  |        |  |  |  |  |  |
| 5 |        |  |        |  |  |  |  |  |
| 6 |        |  |        |  |  |  |  |  |

Whoops! I don't know of any function called openFence. Did we define it correctly? I have to stop executing this program because of this error.

**Stop**

### gidget

|              |           |
|--------------|-----------|
| energy       | 94        |
| grabbed      | [ ]       |
| image        | "default" |
| labeled      | true      |
| layer        | 1         |
| name         | "gidget"  |
| position     | [ 6, 5 ]  |
| rotation     | 0         |
| scale        | 1         |
| transparency | 1         |
| code()       | [ ]       |

### button

|              |            |
|--------------|------------|
| energy       | 100        |
| grabbed      | [ ]        |
| image        | "default"  |
| labeled      | true       |
| layer        | 1          |
| name         | "button"   |
| openGate     | openGate() |
| position     | [ 6, 5 ]   |
| rotation     | 0          |
| scale        | 1          |
| transparency | 1          |
| code()       | [ ]        |

Q&A

# Gidget

helpgidget.org Lee et al. 2014, VL/HCC



Mike Lee

Level 20. Press the button, open the gate!

## code

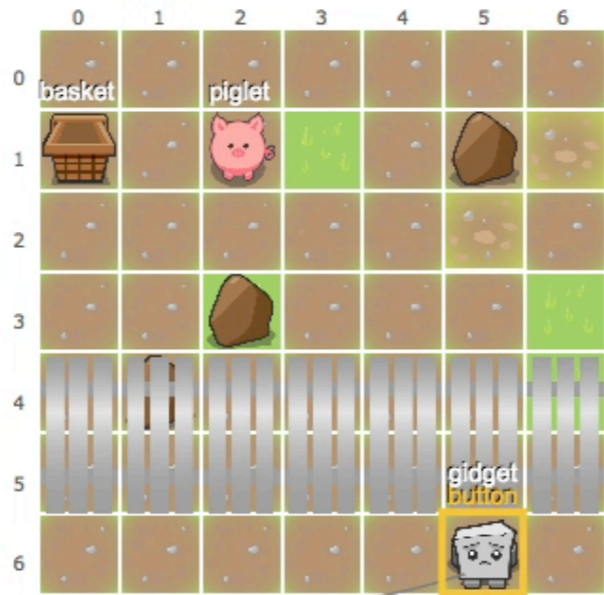
Original Code Clear Code

```
goto /button/  
say "Let's click the button to see its function"  
/button/:openFence()  
function getPiglet() ?  
  goto /piglet/  
  set /piglet/:nickname to "wilbur"  
  set /piglet/:age to 3  
  grab /piglet/  
getBird() ?  
getThePiggy() ?  
goto /basket/
```

```
ensure /piglet/:nickname = "babe"  
ensure /piglet/:age = 3  
ensure # /piglet/ on /basket/ = 1
```

One step One line To end Stop!

## world



Whoops! I don't know of any function called openFence. Did we define it correctly? I have to stop executing this program because of this error.

Stop

## gidget

|              |           |
|--------------|-----------|
| energy       | 94        |
| grabbed      | [ ]       |
| image        | "default" |
| labeled      | true      |
| layer        | 1         |
| name         | "gidget"  |
| position     | [ 6, 5 ]  |
| rotation     | 0         |
| scale        | 1         |
| transparency | 1         |
| code()       | [ ]       |

## button

|              |            |
|--------------|------------|
| energy       | 100        |
| grabbed      | [ ]        |
| image        | "default"  |
| labeled      | true       |
| layer        | 1          |
| name         | "button"   |
| openGate     | openGate() |
| position     | [ 6, 5 ]   |
| rotation     | 0          |
| scale        | 1          |
| transparency | 1          |
| code()       | [ ]        |



# Gidget helpgidget.org



Mike Lee



- Four online controlled experiments with over 1,000 adult learners:
  - Learning is **2x as fast** as Codecademy tutorial, **2x as much** as open-ended creative exploration Lee & Ko 2015
  - **Assessment levels** significantly increase learning efficiency Lee et al. 2013
  - Giving compiler a face and using personal pronouns (I, you, we) draws learner's attention to semantics, **doubling learning efficiency** Lee & Ko 2011
  - Changes attitudes about difficulty of learning to code from negative to positive in **20 minutes** Charters et al. 2014

# Gidget helpgidget.org



Mike Lee



- Four online controlled experiments with over 1,000 adult learners:
  - Learning is **2x as fast** as Codecademy tutorial, **2x as much** as open-ended creative exploration Lee & Ko 2015
  - **Assessment levels** significantly increase learning efficiency Lee et al. 2013
  - Giving compiler a face and using personal pronouns (I, you, we) draws learner's attention to semantics, **doubling learning efficiency** Lee & Ko 2011
  - Changes attitudes about difficulty of learning to code from negative to positive in **20 minutes** Charters et al. 2014

# Gidget

helpgidget.org Lee et al. 2014, VL/HCC

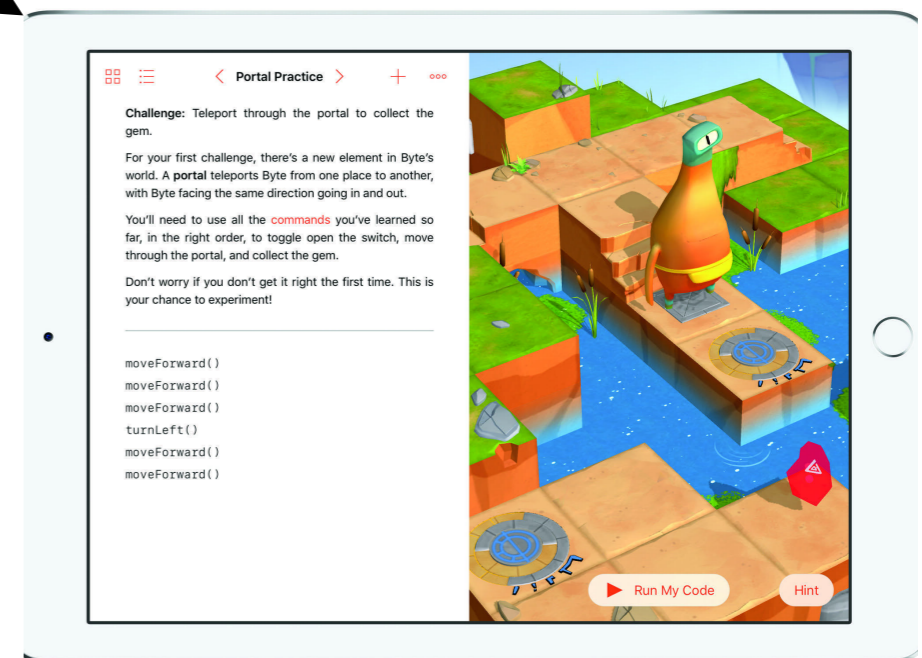


Mike Lee



20,000+ have played via word of mouth, including Chicago Public Schools, many retirees (apparently including my mom)

Directly impacted the design of [code.org](http://code.org)'s CodeStudio and Apple's Swift Playgrounds, used by 10+ million learners.



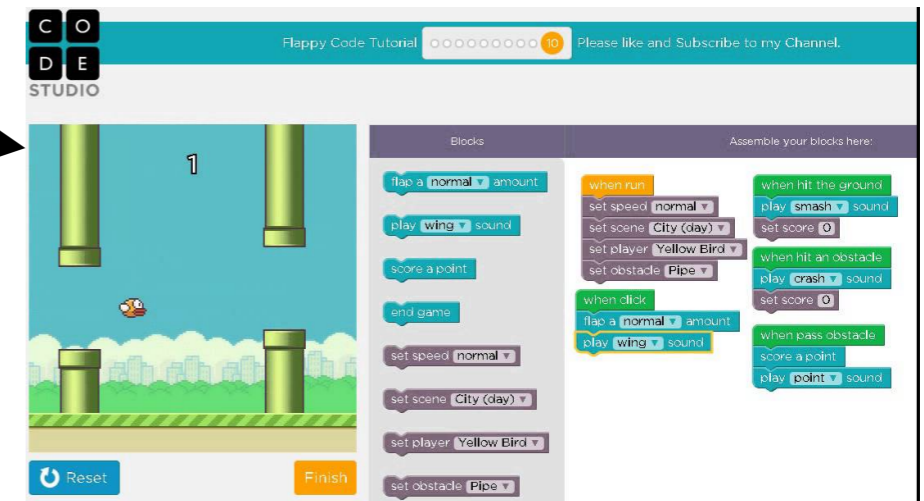


# Gidget

helpgidget.org Lee et al. 2014, VL/HCC

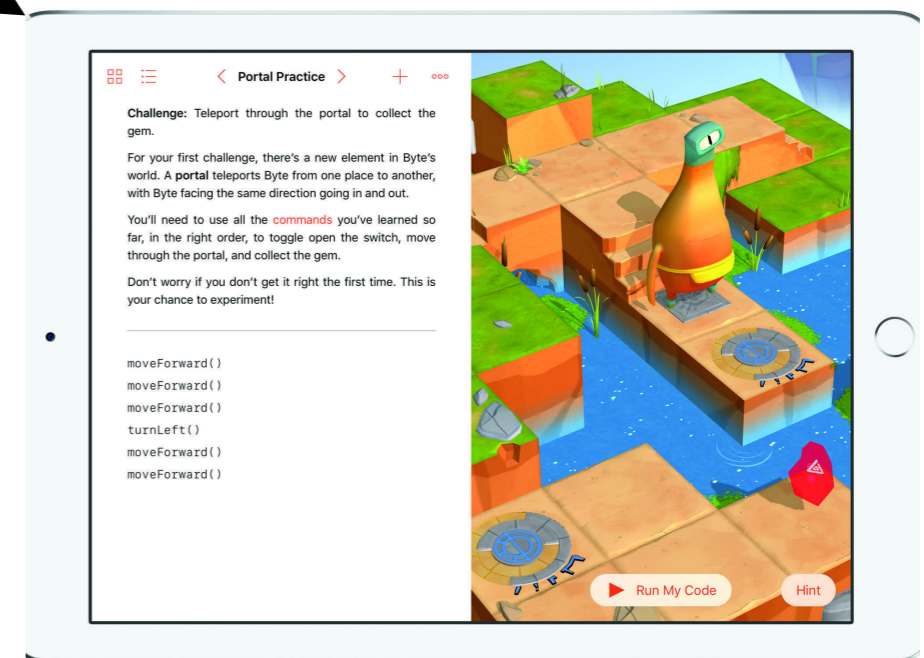


Mike Lee



20,000+ have played via word of mouth, including Chicago Public Schools, many retirees (apparently including my mom)

Directly impacted the design of [code.org](http://code.org)'s CodeStudio and Apple's Swift Playgrounds, used by 10+ million learners.



# Three ideas

## Gidget



Mike Lee, Ph.D.

Learners discover  
semantics through  
debugging

## PLTutor



Greg Nelson

Tutor explicitly  
*teaches*  
semantics

## Tracing Strategies



Benji Xie

Learner  
reminded to  
follow semantics

# Three ideas

## Gidget



Mike Lee, Ph.D.

Learners *discover*  
semantics through  
debugging

## PLTutor



Greg Nelson

Tutor explicitly  
*teaches*  
semantics

## Tracing Strategies



Benji Xie

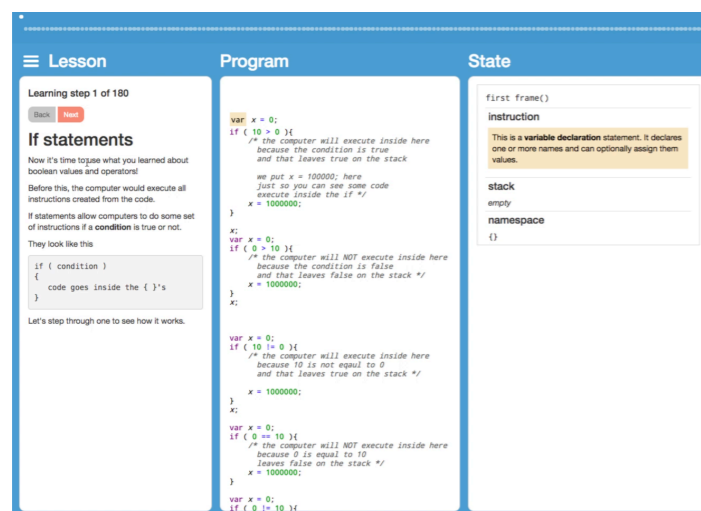
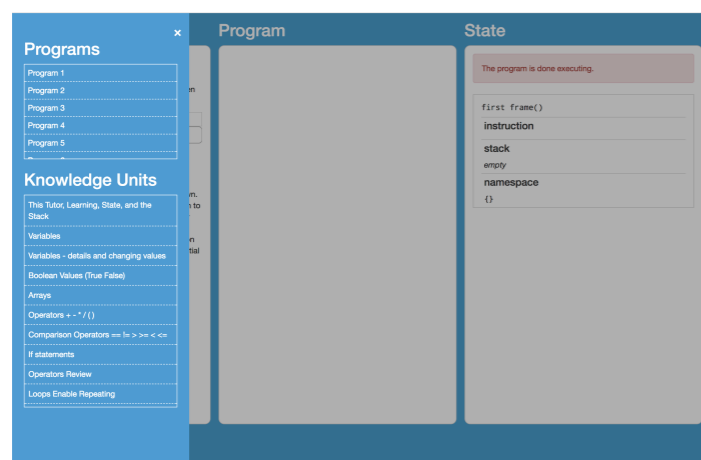
Learner  
reminded to  
follow semantics

# PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017).  
Comprehension First: Evaluating a Novel Pedagogy and Tutoring  
System for Program Tracing in CS1. ACM ICER.



Greg Nelson



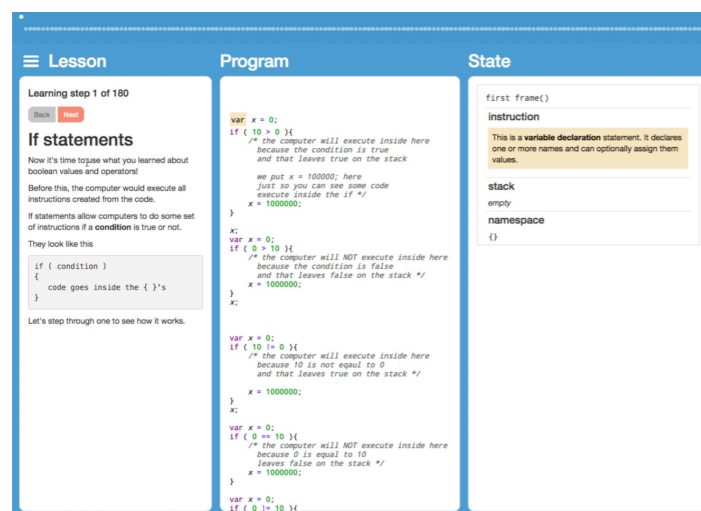
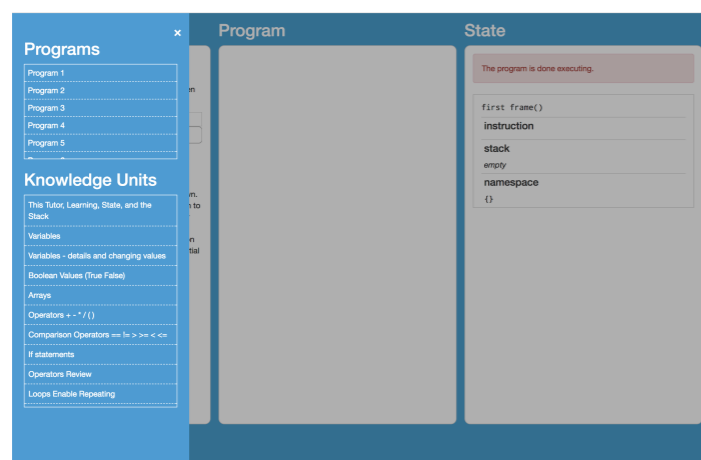
- Convert operational semantics into an **interactive textbook** to be read before learning to write programs
- Covers the entire *JavaScript* semantics in about **3 hours** of practice
- Learner should be able to accurately predict the behavior of *any* JavaScript program

# PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017).  
Comprehension First: Evaluating a Novel Pedagogy and Tutoring  
System for Program Tracing in CS1. ACM ICER.



Greg Nelson



- Convert operational semantics into an **interactive textbook** to be read before learning to write programs
- Covers the entire *JavaScript* semantics in about **3 hours** of practice
- Learner should be able to accurately predict the behavior of *any* JavaScript program

# PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017).  
Comprehension First: Evaluating a Novel Pedagogy and Tutoring  
System for Program Tracing in CS1. ACM ICER.



Greg Nelson

Each chapter covers a set of semantics through a series of programs

The screenshot displays the PLTutor interface with four main panels:

- Programs**: A list of five programs (Program 1 to Program 5).
- Knowledge Units**: A list of learning units including "This Tutor, Learning, State, and the Stack", "Variables", "Variables - details and changing values", "Boolean Values (True False)", "Arrays", "Operators + - \* / ()", "Comparison Operators == != > >= < <=", and "If statements".
- Program**: A large empty area for displaying the current program's code.
- State**: A panel showing the execution state. It contains a message "The program is done executing." and a stack trace for "first frame()" with fields for "instruction", "stack" (empty), and "namespace" ({}).

# PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017).  
Comprehension First: Evaluating a Novel Pedagogy and Tutoring System for Program Tracing in CS1. ACM ICER.



Greg Nelson

**Lesson** explains purpose of semantics

**Program** links syntax to semantics

**State** explains semantics

The screenshot shows the PLTutor interface with three main panels: Lesson, Program, and State.

- Lesson Panel:** Shows "Learning step 1 of 180" with "Back" and "Next" buttons. The title is "If statements". The text explains that it's time to use boolean values and operators, and that if statements allow computers to execute code based on a condition. It includes a code snippet: 

```
if ( condition )  
{  
  code goes inside the { }'s  
}
```
- Program Panel:** Shows two code snippets. The first is: 

```
var x = 0;  
if ( 10 > 0 ){  
  /* the computer will execute inside here  
  because the condition is true  
  and that leaves true on the stack  
  
  we put x = 100000; here  
  just so you can see some code  
  execute inside the if */  
  x = 1000000;  
}
```

 The second is: 

```
x;  
var x = 0;  
if ( 0 > 10 ){  
  /* the computer will NOT execute inside here  
  because the condition is false  
  and that leaves false on the stack */  
  x = 1000000;  
}
```
- State Panel:** Shows the state of the program. It includes a "first frame()" section with an "instruction" box containing the text: "This is a **variable declaration** statement. It declares one or more names and can optionally assign them values." Below this, it shows the "stack" as "empty" and the "namespace" as "{}".

# PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017).  
Comprehension First: Evaluating a Novel Pedagogy and Tutoring System for Program Tracing in CS1. ACM ICER.



Greg Nelson

**Lesson** explains purpose of semantics

**Program** links syntax to semantics

**State** explains semantics

The screenshot shows the PLTutor interface with three main panels: Lesson, Program, and State.

- Lesson Panel:** Shows "Learning step 1 of 180" with "Back" and "Next" buttons. The title is "If statements". The text explains that it's time to use what was learned about boolean values and operators, and that if statements allow computers to execute code only if a condition is true. It includes a code snippet: 

```
if ( condition )  
{  
  code goes inside the { }'s  
}
```
- Program Panel:** Shows two code snippets. The first is: 

```
var x = 0;  
if ( 10 > 0 ){  
  /* the computer will execute inside here  
  because the condition is true  
  and that leaves true on the stack  
  
  we put x = 100000; here  
  just so you can see some code  
  execute inside the if */  
  x = 1000000;  
}
```

 The second is: 

```
x;  
var x = 0;  
if ( 0 > 10 ){  
  /* the computer will NOT execute inside here  
  because the condition is false  
  and that leaves false on the stack */  
  x = 1000000;  
}
```
- State Panel:** Shows the state of the program. It includes a "first frame()" section with an "instruction" box containing the text: "This is a **variable declaration** statement. It declares one or more names and can optionally assign them values." Below this, it shows the "stack" as "empty" and the "namespace" as "{}".



# PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017).  
Comprehension First: Evaluating a Novel Pedagogy and Tutoring  
System for Program Tracing in CS1. ACM ICER.



Greg Nelson

purpose of  
semantics

## Lesson

Learning step 1 of 180

Back Next

### If statements

Now it's time to use what you learned about boolean values and operators!

Before this, the computer would execute all instructions created from the code.

If statements allow computers to do some set of instructions if a **condition** is true or not.

They look like this

```
if ( condition )
{
    code goes inside the { }'s
}
```

Let's step through one to see how it works.

## Program

```
var x = 0;
if ( 10 > 0 ){
    /* the computer will execute inside here
    because the condition is true
    and that leaves true on the stack

    we put x = 100000; here
    just so you can see some code
    execute inside the if */
    x = 100000;
}

x;
var x = 0;
if ( 0 > 10 ){
    /* the computer will NOT execute inside here
    because the condition is false
    and that leaves false on the stack */
    x = 100000;
}

x;
```

```
var x = 0;
if ( 10 != 0 ){
```

# PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017).  
Comprehension First: Evaluating a Novel Pedagogy and Tutoring  
System for Program Tracing in CS1. ACM ICER.



Greg Nelson

purpose of  
semantics

The screenshot shows the PLTutor interface with a blue header. On the left, a 'Lesson' panel contains the text: 'Learning step 1 of 180', 'Back' and 'Next' buttons, the title 'If statements', and explanatory text about boolean values and operators. A code block shows the general syntax of an if statement. On the right, a 'Program' panel shows two code snippets. The first snippet shows an if statement with a true condition (10 > 0) and a comment explaining that the code inside will execute. The second snippet shows an if statement with a false condition (0 > 10) and a comment explaining that the code inside will not execute. A red diamond icon is in the bottom right corner.

Lesson

Learning step 1 of 180

Back Next

## If statements

Now it's time to use what you learned about boolean values and operators!

Before this, the computer would execute all instructions created from the code.

If statements allow computers to do some set of instructions if a **condition** is true or not.

They look like this

```
if ( condition )
{
    code goes inside the { }'s
}
```

Let's step through one to see how it works.

Program

```
var x = 0;
if ( 10 > 0 ){
    /* the computer will execute inside here
    because the condition is true
    and that leaves true on the stack

    we put x = 100000; here
    just so you can see some code
    execute inside the if */
    x = 100000;
}

x;
var x = 0;
if ( 0 > 10 ){
    /* the computer will NOT execute inside here
    because the condition is false
    and that leaves false on the stack */
    x = 100000;
}

x;

var x = 0;
if ( 10 != 0 ){
```

# PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017).  
Comprehension First: Evaluating a Novel Pedagogy and Tutoring  
System for Program Tracing in CS1. ACM ICER.



Greg Nelson

**Next** moves through both program execution trace *and* instruction.

The screenshot shows the PLTutor interface with two main panels: 'Lesson' and 'Program'. The 'Lesson' panel is on the left and contains the following text:

Learning step 1 of 180

Back Next

## If statements

Now it's time to use what you learned about boolean values and operators!

Before this, the computer would execute all instructions created from the code.

If statements allow computers to do some set

The 'Program' panel is on the right and contains the following code snippet:

```
var x = 0;
if ( 10 > 0 ){
    /* the computer will execute inside here
       because the condition is true
       and that leaves true on the stack

       we put x = 100000; here
       just so you can see some code
       execute inside the if */
    x = 100000;
}
```

# PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017).  
Comprehension First: Evaluating a Novel Pedagogy and Tutoring  
System for Program Tracing in CS1. ACM ICER.



Greg Nelson

**Next** moves through both program execution trace *and* instruction.

The screenshot shows the PLTutor interface with two main panels: 'Lesson' and 'Program'. The 'Lesson' panel is on the left and contains the following text:

Learning step 1 of 180

Back Next

## If statements

Now it's time to use what you learned about boolean values and operators!

Before this, the computer would execute all instructions created from the code.

If statements allow computers to do some set

The 'Program' panel is on the right and contains the following code snippet:

```
var x = 0;
if ( 10 > 0 ){
    /* the computer will execute inside here
       because the condition is true
       and that leaves true on the stack

       we put x = 100000; here
       just so you can see some code
       execute inside the if */
    x = 100000;
}
```



# PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017).  
Comprehension First: Evaluating a Novel Pedagogy and Tutoring  
System for Program Tracing in CS1. ACM ICER.



Greg Nelson

*State* teaches semantics in a runtime context,  
*Lesson* generalizes back to purpose

The screenshot displays the PLTutor interface with three main panels: Lesson, Program, and State.

- Lesson Panel:** Shows "Learning step 7 of 180" with "Back" and "Next" buttons. The text reads: "Carefully step through the code and read all the explanations for the instructions, and look at the stack."
- Program Panel:** Displays code with annotations. The first code block is: 

```
var x = 0;
if ( 10 > 0 ){
  /* the computer will execute inside here
  because the condition is true
  and that leaves true on the stack

  we put x = 100000; here
  just so you can see some code
  execute inside the if */
  x = 1000000;
}
```

 The second code block is: 

```
x;
var x = 0;
if ( 0 > 10 ){
  /* the computer will NOT execute inside here
  because the condition is false
  and that leaves false on the stack */
  x = 1000000;
}
x;
```
- State Panel:** Shows the state of the first frame(). It includes an "instruction" box with the text: "Before moving to the next statement, we remove the value this expression left on the stack, if any." Below this, it shows "stack" as "empty" and "namespace" as a block containing a variable "x" with the value "0".

# PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017).  
Comprehension First: Evaluating a Novel Pedagogy and Tutoring  
System for Program Tracing in CS1. ACM ICER.



Greg Nelson

*State* teaches semantics in a runtime context,  
*Lesson* generalizes back to purpose

The screenshot displays the PLTutor interface with three main panels: Lesson, Program, and State.

- Lesson Panel:** Shows "Learning step 7 of 180" with "Back" and "Next" buttons. The instruction reads: "Carefully step through the code and read all the explanations for the instructions, and look at the stack."
- Program Panel:** Displays code with annotations. The first code block shows a variable `x` being set to 0, followed by an `if (10 > 0)` block. An annotation explains that the computer will execute inside the `if` block because the condition is true, and that the value `true` is left on the stack. Below this, it notes that `x = 100000` is executed. The second code block shows `x` being accessed, followed by `var x = 0;` and `if (0 > 10)`. An annotation explains that the computer will NOT execute inside the `if` block because the condition is false, and that the value `false` is left on the stack. Finally, `x = 100000;` is executed.
- State Panel:** Shows the current state of the program. It includes a "first frame()" section, an "instruction" section with a yellow highlight explaining that the value of the expression left on the stack is removed before moving to the next statement, a "stack" section showing it is empty, and a "namespace" section showing a variable `x` with the value `0`.



# PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017).  
Comprehension First: Evaluating a Novel Pedagogy and Tutoring  
System for Program Tracing in CS1. ACM ICER.



Greg Nelson

Reverse execution allows learner to review instruction they didn't understand.

The screenshot displays the PLTutor interface with three main panels:

- Lesson:** Shows "Learning step 15 of 180" with "Back" and "Next" buttons. A large blue arrow icon is labeled "View Program".
- Program:** Displays code snippets with annotations. The first snippet shows an if-statement where the condition is true, and the code inside is executed, setting `x = 1000000`. The second snippet shows the same if-statement but with a false condition, so the code inside is not executed. The third snippet shows an if-statement with a condition `10 != 0`, which is true.
- State:** Shows the current state of the program. It includes a "first frame()" section, an "instruction" box with the text "Assign the value on top of the stack to x", a "stack" box containing the value "1000000", and a "namespace" box containing a variable `x` with the value "0".

# PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017).  
Comprehension First: Evaluating a Novel Pedagogy and Tutoring  
System for Program Tracing in CS1. ACM ICER.



Greg Nelson

Reverse execution allows learner to review instruction they didn't understand.

The screenshot displays the PLTutor interface with three main panels:

- Lesson:** Shows "Learning step 15 of 180" with "Back" and "Next" buttons. A large blue arrow icon is labeled "View Program".
- Program:** Displays code with annotations. The first code block is highlighted in yellow and includes the comment: */\* the computer will execute inside here because the condition is true and that leaves true on the stack we put x = 100000; here just so you can see some code execute inside the if \*/*. The code is: 

```
var x = 0;
if ( 10 > 0 ){
  x = 1000000;
}
x;
```
- State:** Shows the execution state for "first frame()". It includes an "instruction" box with the text "Assign the value on top of the stack to x", a "stack" box containing the value "1000000", and a "namespace" box containing a variable "x" with the value "0".





# PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017).  
Comprehension First: Evaluating a Novel Pedagogy and Tutoring  
System for Program Tracing in CS1. ACM ICER.



Greg Nelson

**Lesson** explains the side effect of the semantics before proceeding to further examples.

A screenshot of the PLTutor interface. The top navigation bar is blue with a white hamburger menu icon and the word "Lesson" in white. Below the navigation bar, there is a white card with a blue border. The card contains the text "Learning step 18 of 180" in bold. Below this text are two buttons: a red "Back" button and a grey "Next" button. A mouse cursor is hovering over the "Next" button. Below the buttons, there is a paragraph of text: "x has a new value now because the condition 10 &gt; 0 was true. You can see it on the stack." To the right of the main content area, there is a partial view of a code editor showing the text "var" and "if (" in purple. At the bottom of the page, the number "57" is displayed.

# PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017).  
Comprehension First: Evaluating a Novel Pedagogy and Tutoring  
System for Program Tracing in CS1. ACM ICER.



Greg Nelson

**Lesson** explains the side effect of the semantics before proceeding to further examples.

A screenshot of the PLTutor interface. The top navigation bar is blue with a white hamburger menu icon and the word "Lesson" in white. Below the navigation bar, there is a white card with a blue border. The card contains the text "Learning step 18 of 180" in bold. Below this text are two buttons: "Back" (red) and "Next" (grey). A mouse cursor is hovering over the "Next" button. Below the buttons, there is a paragraph of text: "x has a new value now because the condition 10 &gt; 0 was true. You can see it on the stack." To the right of the main content area, there is a partial view of a code editor with the text "var" and "if (" visible. At the bottom of the page, the number "57" is displayed.

# PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017).  
Comprehension First: Evaluating a Novel Pedagogy and Tutoring  
System for Program Tracing in CS1. ACM ICER.



Greg Nelson

**Assessments** embedded in execution trace require learners to predict side effects of semantics.

The screenshot shows the PLTutor interface with three main panels: Lesson, Program, and State.

- Lesson Panel:** Shows "Learning step 29 of 180" with "Back" and "Next" buttons. A question asks "What did `0 > 10` evaluate to (leave on the stack)?" with four answer buttons: "true", "10", "-10", and "false".
- Program Panel:** Displays code snippets with annotations. The first snippet shows an `if (10 > 0)` block where the condition is true, and the code inside is `x = 1000000;`. The second snippet shows an `if (0 > 10)` block where the condition is false, and the code inside is `x = 1000000;`. The third snippet shows an `if (10 != 0)` block where the condition is true, and the code inside is `x = 1000000;`.
- State Panel:** Shows the current state of the program. It includes a "first frame()" section with an "instruction" box containing the text: "If the if statement's condition is true, execute the true statements. Otherwise, skip it." Below this is a "stack" section with a question mark, and a "namespace" section showing a variable `x` with the value `0`.

# PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017).  
Comprehension First: Evaluating a Novel Pedagogy and Tutoring  
System for Program Tracing in CS1. ACM ICER.



Greg Nelson

**Assessments** embedded in execution trace require learners to predict side effects of semantics.

The screenshot shows the PLTutor interface with three main panels: Lesson, Program, and State.

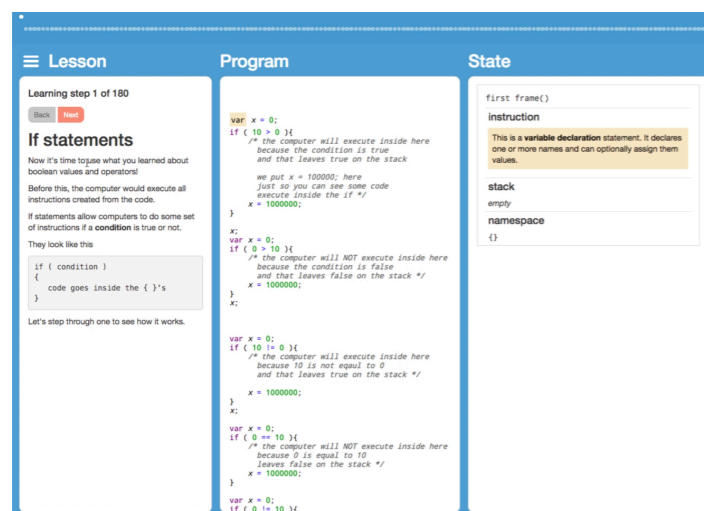
- Lesson Panel:** Shows "Learning step 29 of 180" with "Back" and "Next" buttons. Below is a question: "What did `0 > 10` evaluate to (leave on the stack)?" with four answer buttons: "true", "10", "-10", and "false".
- Program Panel:** Displays code snippets with annotations. The first snippet shows an `if (10 > 0)` block with a comment: */\* the computer will execute inside here because the condition is true and that leaves true on the stack \*/*. Below it, `x = 1000000;` is shown. The second snippet shows an `if (0 > 10)` block with a comment: */\* the computer will NOT execute inside here because the condition is false and that leaves false on the stack \*/*. Below it, `x = 1000000;` is shown. A third snippet shows `if (10 != 0)` with a comment: */\* the computer will execute inside here because 10 is not equal to 0 and that leaves true on the stack \*/*.
- State Panel:** Shows the current state of the program. It includes a "first frame()" section with an "instruction" box containing the text: "If the if statement's condition is true, execute the true statements. Otherwise, skip it." Below this is a "stack" section with a box containing a question mark "?". At the bottom is a "namespace" section showing a variable `x` with the value `0`.

# PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017).  
Comprehension First: Evaluating a Novel Pedagogy and Tutoring  
System for Program Tracing in CS1. ACM ICER.



Greg Nelson



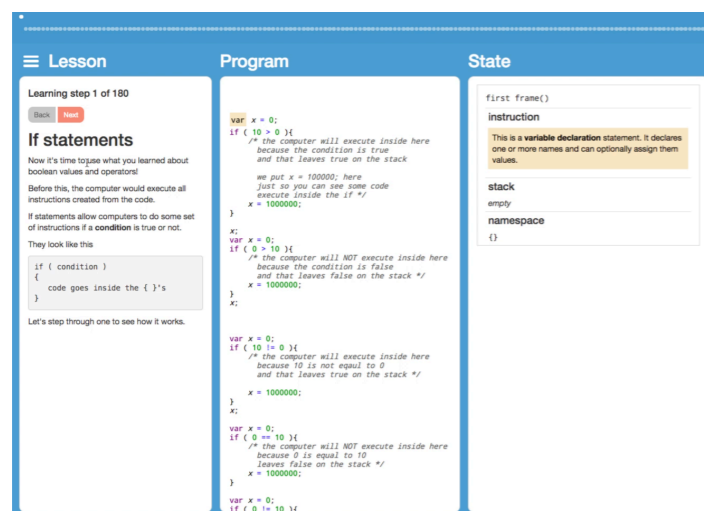
- Required complete re-architecting of language stack
- Must preserve provenance of all compiler and runtime state to facilitate reversibility and embedded explanations
- Redesigned grammar to facilitate granular explanations

# PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017).  
Comprehension First: Evaluating a Novel Pedagogy and Tutoring  
System for Program Tracing in CS1. ACM ICER.



Greg Nelson



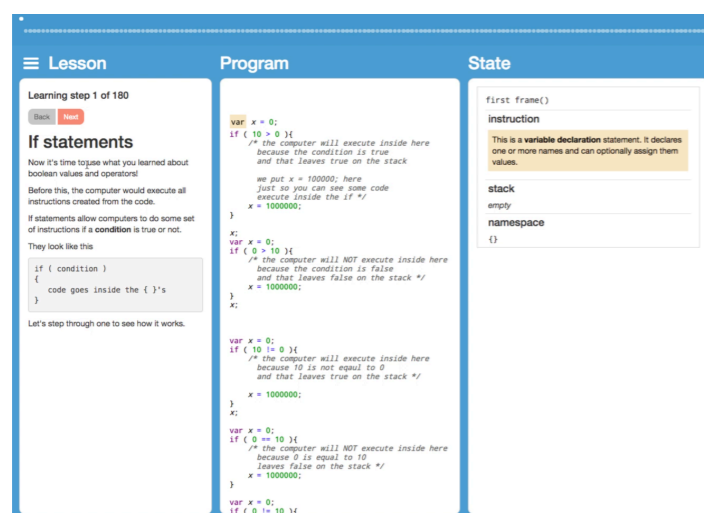
- Required complete re-architecting of language stack
- Must preserve provenance of all compiler and runtime state to facilitate reversibility and embedded explanations
- Redesigned grammar to facilitate granular explanations

# PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017).  
Comprehension First: Evaluating a Novel Pedagogy and Tutoring  
System for Program Tracing in CS1. ACM ICER.



Greg Nelson



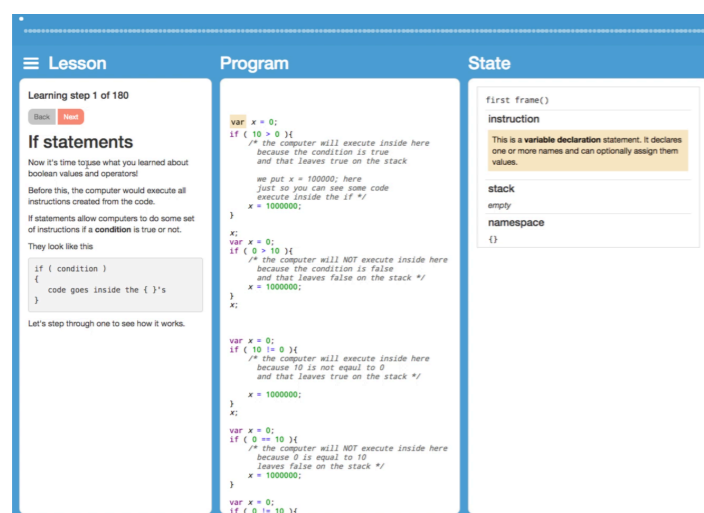
- Compared PLTutor to Codecademy in a 4-hour controlled experiment with **40 CS1** students
- Measured learning with SCS1, a validated assessment of CS1 learning
- PLTutor had **60% higher** learning gains, learning gains predicted midterm scores

# PLTutor

Greg Nelson, Benjamin Xie, and Andrew J. Ko (2017).  
Comprehension First: Evaluating a Novel Pedagogy and Tutoring  
System for Program Tracing in CS1. ACM ICER.



Greg Nelson



- Compared PLTutor to Codecademy in a 4-hour controlled experiment with **40 CS1** students
- Measured learning with SCS1, a validated assessment of CS1 learning
- PLTutor had **60% higher** learning gains, learning gains predicted midterm scores



# Three ideas

## Gidget



Mike Lee, Ph.D.

Learners *discover*  
semantics through  
debugging

## PLTutor



Greg Nelson

Tutor explicitly  
*teaches*  
semantics

## Tracing Strategies



Benji Xie

Learner  
reminded to  
follow semantics

# Three ideas

## Gidget



Mike Lee, Ph.D.

Learners *discover* semantics through debugging

## PLTutor



Greg Nelson

Tutor explicitly teaches semantics

## Tracing Strategies



Benji Xie

Learner reminded to follow semantics

# Strategy

Benjamin Xie, Greg Nelson, and Andrew J. Ko (2017). An Explicit Strategy to Scaffold Novice Program Tracing. SIGCSE.



Benji Xie

STRATEGY: Understanding the Problem;  
Run the Code (like a computer).

**UNDERSTAND THE PROBLEM**

1. Read question: Understand what you are being asked to do. At the end of the problem instructions, write a check mark: ✓
2. Find where the program begins executing. At the start of that line, draw an arrow: →

**RUN THE CODE**

3. Execute each line according to the rules of Java:
  - a. From the syntax, determine the rule for each part of the line.
  - b. Follow the rules.
  - c. Update memory table(s).
  - d. Find the code for the next part.
  - e. Repeat until the program terminates.

- When learners have brittle knowledge of semantics, they often **guess** how programs will execute
- An explicit strategy for reading programs should outperform guessing

# Strategy

Benjamin Xie, Greg Nelson, and Andrew J. Ko (2017). An Explicit Strategy to Scaffold Novice Program Tracing. SIGCSE.



Benji Xie

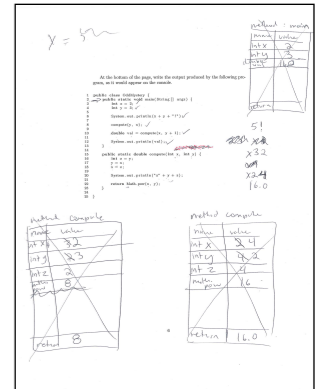
**STRATEGY: Understanding the Problem;  
Run the Code (like a computer).**

## UNDERSTAND THE PROBLEM

1. Read question: Understand what you are being asked to do. At the end of the problem instructions, write a check mark: ✓
2. **Find where the program begins executing.** At the start of that line, draw an arrow: →

## RUN THE CODE

3. Execute each line according to the rules of Java:
  - a. From the syntax, determine the rule for each part of the line.
  - b. Follow the rules.
  - c. Update memory table(s).
  - d. Find the code for the next part.
  - e. Repeat until the program terminates.



# Strategy

Benjamin Xie, Greg Nelson, and Andrew J. Ko (2017). An Explicit Strategy to Scaffold Novice Program Tracing. SIGCSE.



Benji Xie

STRATEGY: Understanding the Problem;  
Run the Code (like a computer).

UNDERSTAND THE PROBLEM

1. Read question. Understand what you are being asked to do. At the end of the problem instructions, write a check mark. ✓
2. Find where the program begins executing. At the start of that line, draw an arrow. →

RUN THE CODE

3. Execute each line according to the rules of Java.
  - a. From the syntax, determine the rule for each part of the line.
  - b. Follow the rules.
  - c. Update memory tables.
  - d. Find the code for the next part.
  - e. Repeat until the program terminates.



method = main

| name       | value |
|------------|-------|
| int x      | 2     |
| int y      | 3     |
| double val | 16.0  |
| return     |       |

At the bottom of the page, write the output produced by the following program, as it would appear on the console.

```
1 public class OddMystery {
2   → public static void main(String[] args) {
3     int x = 2; ✓
4     int y = 3; ✓
5
6     System.out.println(x + y + "!"); ✓
7
8     compute(y, x); ✓
9
10    double val = compute(x, y + 1); ✓
11
12    System.out.println(val); ✓
13  }
14
15  public static double compute(int x, int y) {
16    int z = y;
17    y = x;
18    x = z;
19
20    System.out.println("x" + y + z);
21
22    return Math.pow(x, y);
23  }
24
25 }
```

5!  
~~20~~ x 2  
x 3 2  
~~16~~  
x 2 4  
16.0

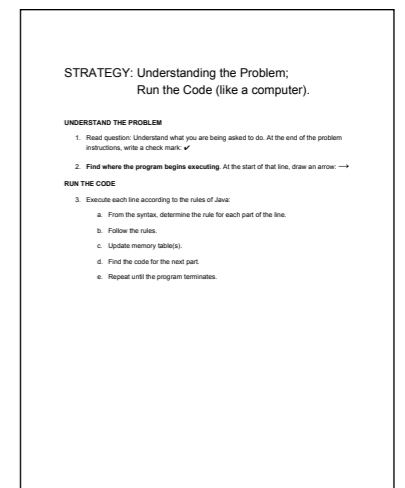
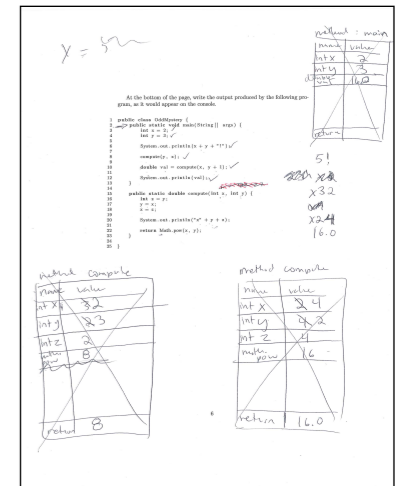
# Strategy

Benjamin Xie, Greg Nelson, and Andrew J. Ko (2017). An Explicit Strategy to Scaffold Novice Program Tracing. SIGCSE.



Benji Xie

- In a controlled experiment with **15 minutes** of practice, 12 students who learned the strategy were more systematic than 12 who didn't, resulting in:
  - **15% higher performance** on problems in the lab
  - **7% higher on midterm** that was mostly writing focused
  - **No midterm failures** (compared to 25% failure in control)



# Making programs easier to *read*

Requiring learners to *directly observe* operational semantics and map them to syntax can significantly increase learning outcomes.

# Making programs easier to write

*One theory, one idea*



# Program writing

- Little prior work theorizing about what **program writing** skills actually are
  - Most prior work compares to expert and novices, showing that novices are unsystematic, speculative, and ineffective
  - A few papers show that the more developers “**self-regulate**” their problem solving, the more productive they are

# Self-regulation

Schraw, Crippen,, & Hartley (2006). Promoting self - regulation in science education. *Research in Science Education* 36, 1-2, 111 - 139.

- From educational psychology, refers to one's ability to reflect on, critique, and control one's thoughts and behaviors during problem solving:
  - Explicit planning skills
  - Explicit monitoring of one's process
  - Explicit monitoring of one's comprehension
  - Reflection on one's cognition
  - Self-explanation of decisions

# Great engineers are highly self-regulating

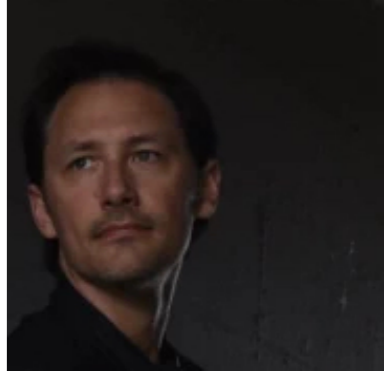
Li, P., Ko, A.J., & Zhu, J. (2015)  
What Makes a Great Software  
Engineer? ICSE.



Paul Li

- Interviewed 59 senior developers at Microsoft and surveyed 1,926 about what makes a *great software engineer*:
- Top attributes included:
  - Resourceful
  - Persistent
  - **Self-regulating**

# Dastyni's theory of program writing

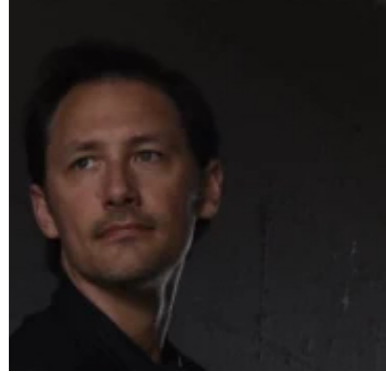


Dastyni Loksa

- Programming involves iteration through **6 key activities**:
  - Interpreting problems
  - Searching for similar problems
  - Searching for solutions
  - Evaluating solutions
  - Implementing solutions
  - Evaluating implementations
- Programming requires:
  - A **knowledge repository** of problems and solutions (in memory or elsewhere)
  - **Self-regulation skills** to help a programmer:
    - Select strategies for completing activity
    - Deciding when a strategy is failing or successful

# Self-regulation is related to success

Loksa, D., Ko, A.J. (2016) . The Role of Self-Regulation in Programming Problem Solving Process and Success. ACM ICER.



Dastyni Loksa

- Observed think aloud of 37 novices in CS1 and CS2 writing solutions to several programming problems.
- Most novices engaged in self-regulation, but infrequently and superficially
- Self-regulation related to fewer errors, but only for novices with sufficient prior knowledge to solve problems

# Can we teach it?

Loksa et al. (2016).  
Programming, Problem Solving,  
and Self-Awareness: Effects of  
Explicit Guidance. ACM CHI.



Dastyni Loksa

- Taught 48 high schoolers with no prior programming experience HTML, CSS, JavaScript and React for 1 week, then had them build personal web sites for 1 more week
- Treatment group received:
  - Learned Dastyni's theory of program writing
  - Before receiving help, required to practice self-regulation, explaining which activity they were doing, what their strategy was, and whether it was working

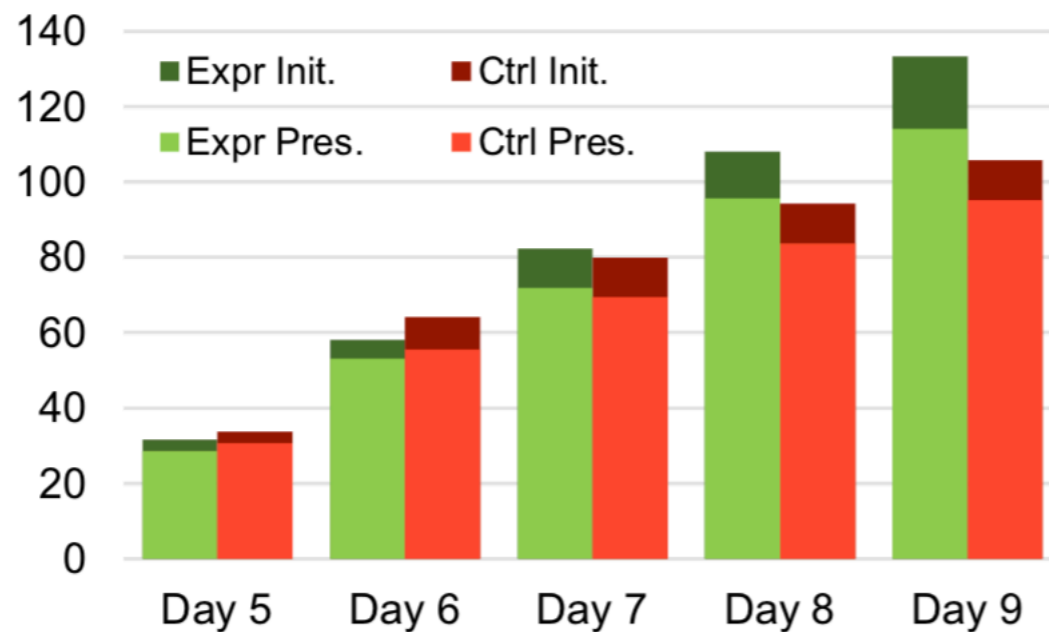
# Yes!

Loksa et al. (2016).  
Programming, Problem Solving,  
and Self-Awareness: Effects of  
Explicit Guidance. ACM CHI.

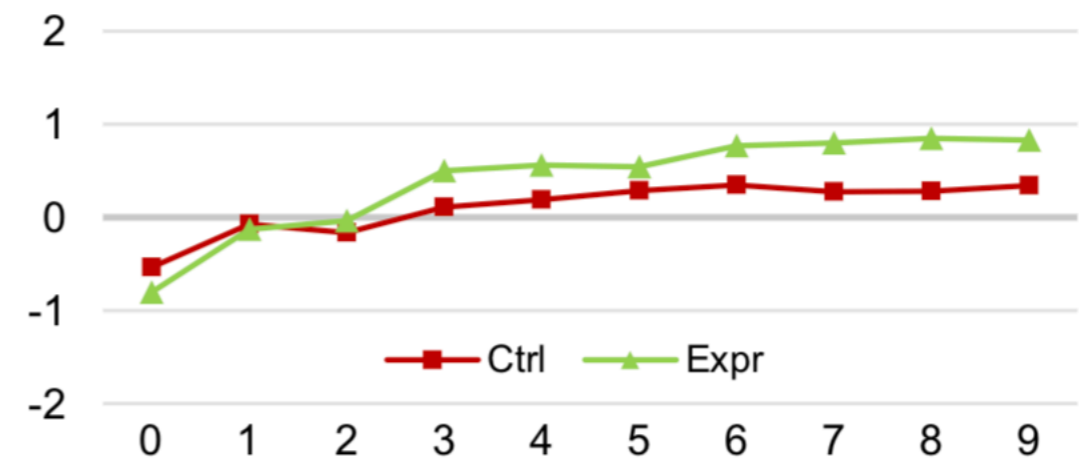


Dastyni Loksa

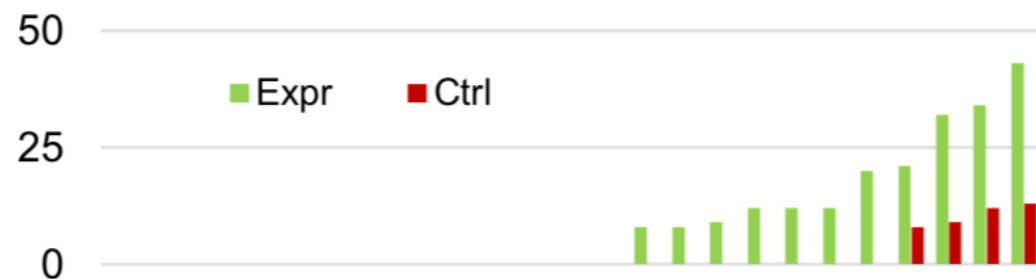
## More productive



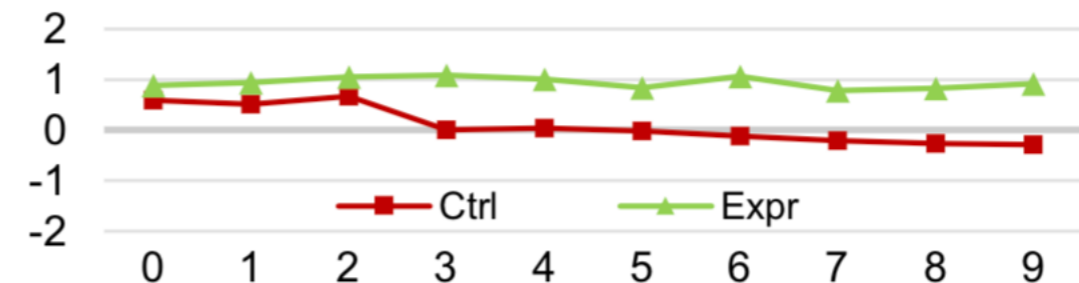
## Higher programming self-efficacy



## More self-defined work



## No growth mindset erosion



# Making programs easier to write

*Teaching programming  
self-regulation promotes  
independence, increased  
productivity, and higher  
self-efficacy.*



# What's next?

# CS1 mastery

New NSF Cyberlearning



Min Li

Benji Xie

- Prior work shows increased learning, but not *mastery*, which requires **personalized content and feedback**
- Human tutors can provide this, but can't scale it
- We're building a tutor that provides **infinite personalized practice** by applying program synthesis and our theories of programming knowledge
- **Goal:** every student masters CS1 content in 10 hours

# Strategies

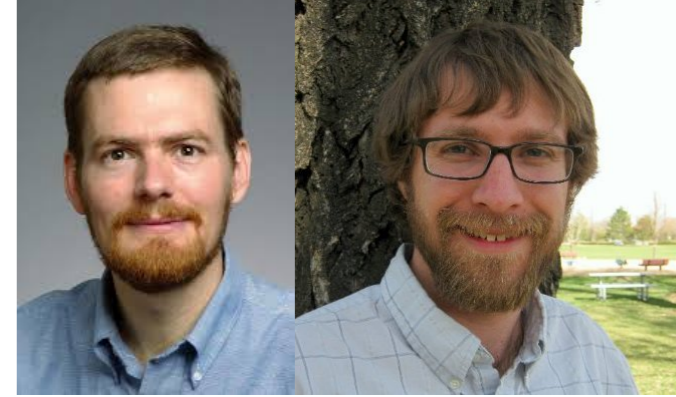
New NSF SHF Medium



Dastyni Loksa Thomas LaToza

- Self-regulation is only useful with good **strategies**
- Defining 1) what programming strategies are, 2) how to describe them, 3) which ones exist, 4) when they're effective, 5) support for learning and executing them.
- **Goal:** A new science of programming strategies analogous to other disciplines' "engineering handbooks," which show how to solve problems in a discipline

# Robust API learning



Mike Ernst Kyle Thayer

- New theory of API knowledge as **domain concepts**, **design templates**, and **API execution semantics**
- Techniques to automatically extract this knowledge from API implementations
- Building a tutor that generates on-demand API tutorials using this extracted knowledge.
- **Goal**: rapid, robust API learning at scale



# Can't do it alone...

- Many great faculty contributing to computing education research from PL, Software Engineering, and HCI. **Join us!**
- Our doctoral students need tenure-track positions to continue their work. **Hire them!**

# Thanks!

Millions try to learn to code, but fail.

Explicit instruction and feedback on semantics is key.

Learning tech like Gidget and PLTutor are scalable and effective

Pedagogies like tracing strategies and self-regulation prompting are effective and immediately adoptable



Supported by NSF, Google, Microsoft, Adobe, the University of Washington.



Thanks to my wonderful doctoral and undergraduate students, and the hundreds of participants in our studies!