Structure/Structured/ Projectional Editors

Quick facts before today's discussion

- CS performance is not bimodal
- Students who use projectional editors vs. textual editors perform equivalently on tests of CS knowledge
- For novices *to a given language* (not necessarily to CS!), projectional editors make users more productive

More on all of these in Thursday's class, but we'll have a more productive discussion today if we go over these before we start chatting :)

Reading Reflection

Discuss in groups

- What do you think is the difference between a visual editor and a projectional editor (if any)?
- Based on the videos:
  - What kinds of errors are projectional editors preventing?
  - What kinds of errors are projectional editors not preventing?
  - If you've ever used a projectional editor (or a projectional editing mode in a textual editor), what did you like or dislike about the experience?

structure editor

==

structured editor

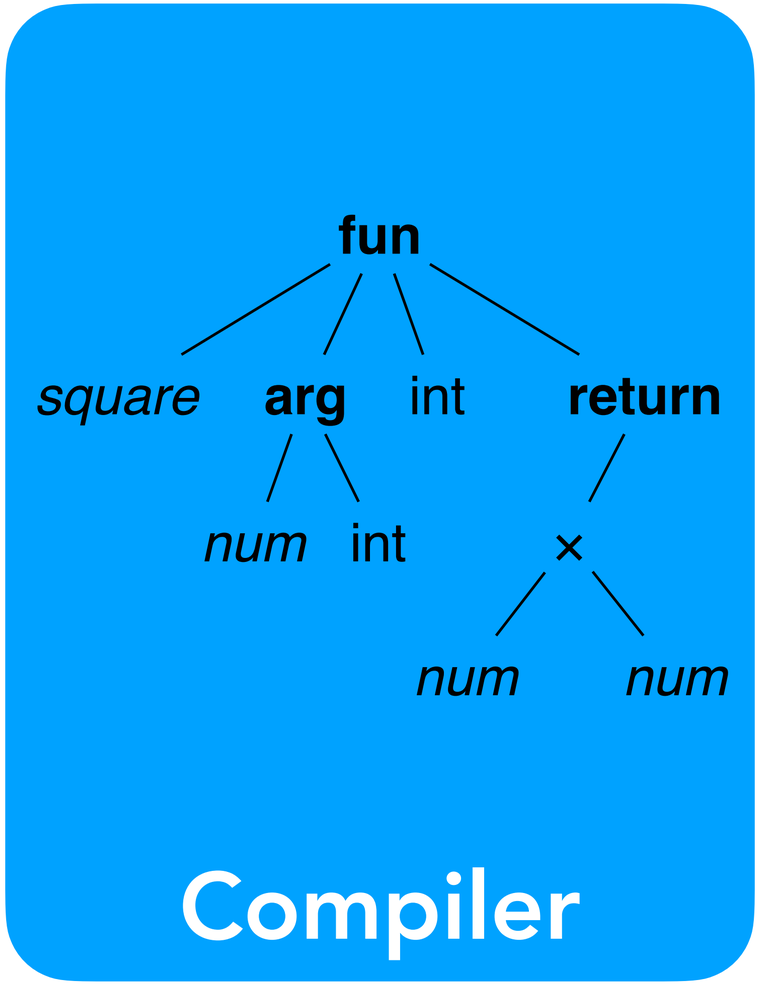
==

projectional editor

What's happening inside my compiler?

```
1   // square a couple numbers
2   int square(int num) {
3       return num * num;
4   }
```

parser



code generator

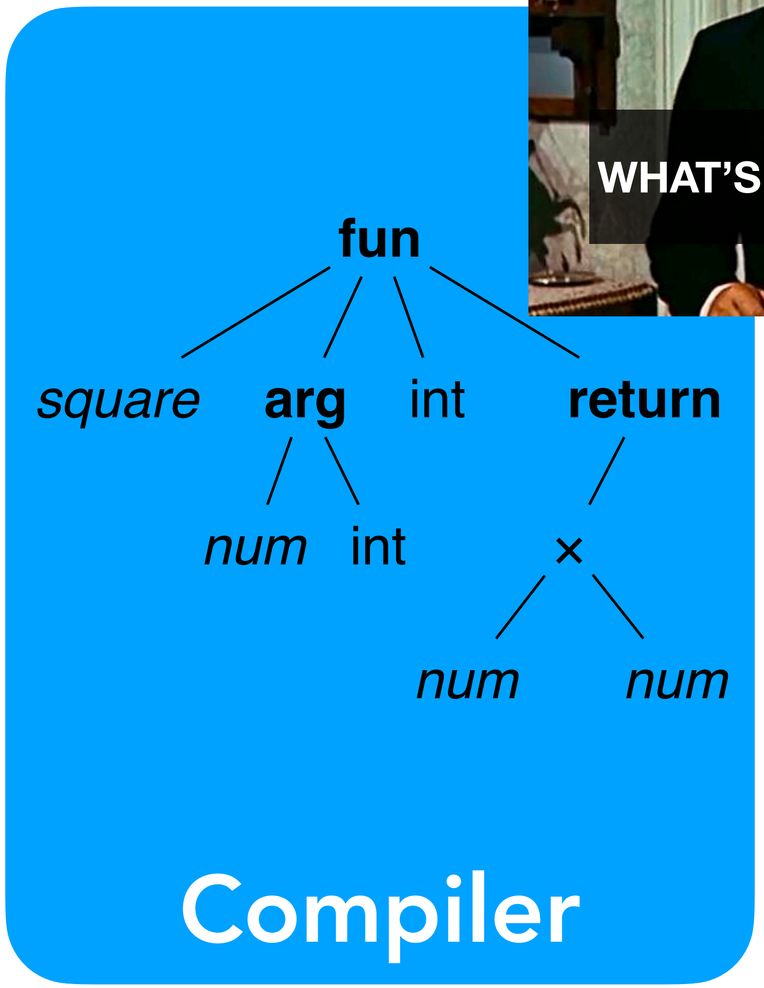
```
1   square(int):
2           push    rbp
3           mov     rbp, rsp
4           mov     DWORD PTR [rbp-4], edi
5           mov     eax, DWORD PTR [rbp-4]
6           imul    eax, eax
7           pop     rbp
8           ret
```

Assembly Language

What's happening inside my compile

parser

```
// square a couple numbers
int square(int num) {
    return num * num;
}
```



erator

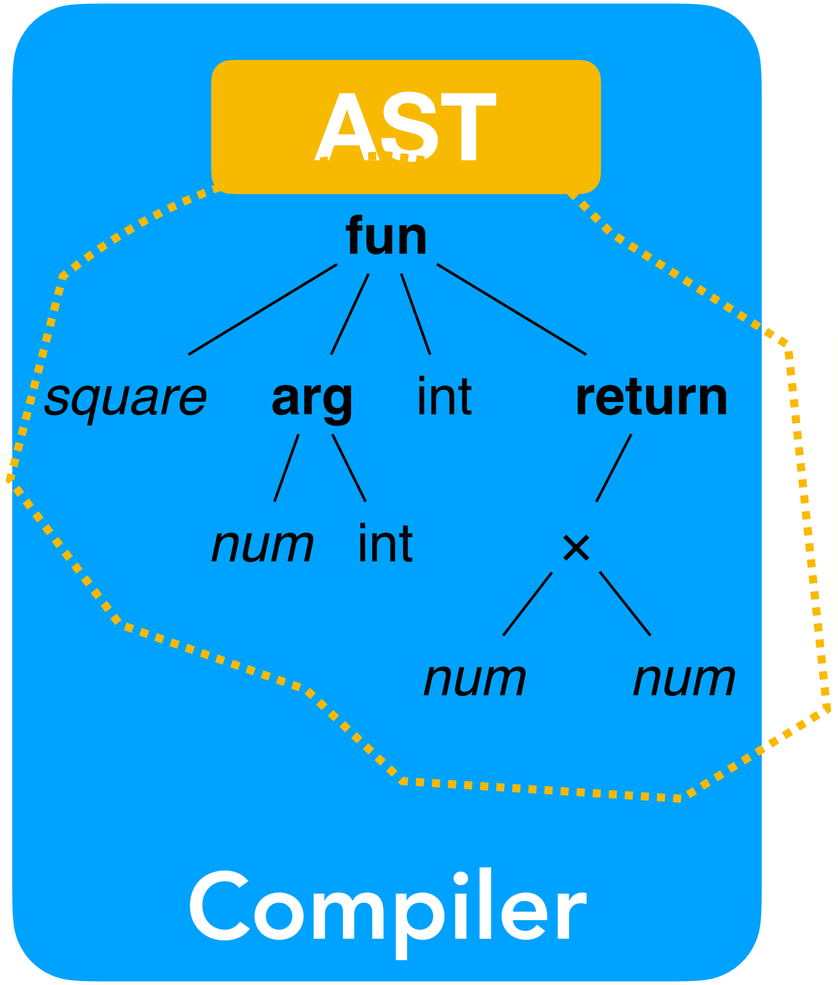
```
square(int):
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-4], edi
        mov     eax, DWORD PTR [rbp-4]
        imul    eax, eax
        pop     rbp
        ret
```

Assembly Language

What's happening inside my compiler?

```
1   // square a couple numbers
2   int square(int num) {
3       return num * num;
4   }
```

parser

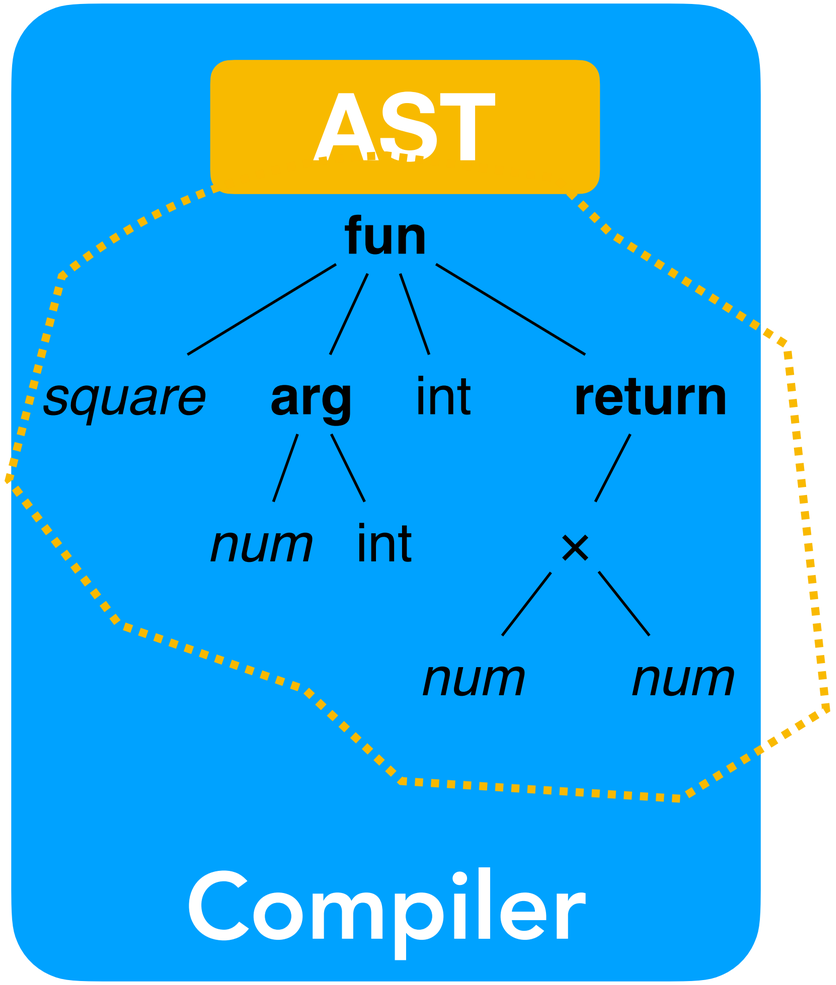


code generator

```
1   square(int):
2           push    rbp
3           mov     rbp, rsp
4           mov     DWORD PTR [rbp-4], edi
5           mov     eax, DWORD PTR [rbp-4]
6           imul    eax, eax
7           pop     rbp
8           ret
```

Assembly Language

Abstract Syntax Tree (AST)

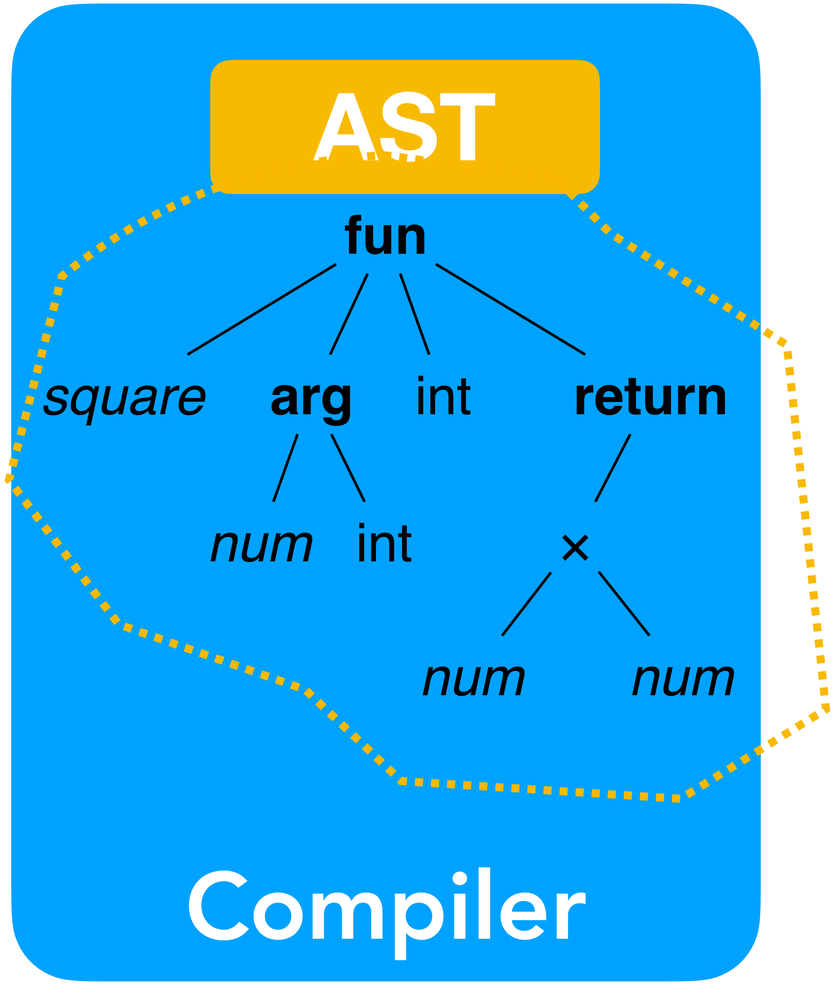


Abstract because we're not putting in every detail of the actual programming language syntax. (E.g., we've dropped all those pesky semicolons and parens.)

Syntax because we're representing the syntactic structure of the code in question.

Tree because…well, obvious. But look, we got to throw away a bunch of parens and other grouping things because it's all in the tree structure now!

Abstract Syntax Tree (AST)



Programs are data! We can mess with them!

…and we can build them up directly. We don't *have* to write in a textual programming language and use a parser to recover this structure.

Projectional Editor

An editor where you're building up the AST directly.

People can argue about the meaning of "directly." How far does it have to be from the actual AST before it stops being a projectional editor? But basically it's just a judgment call.

Projectional *isn't a feature of the programming language*

It's a feature of the programming *environment*!

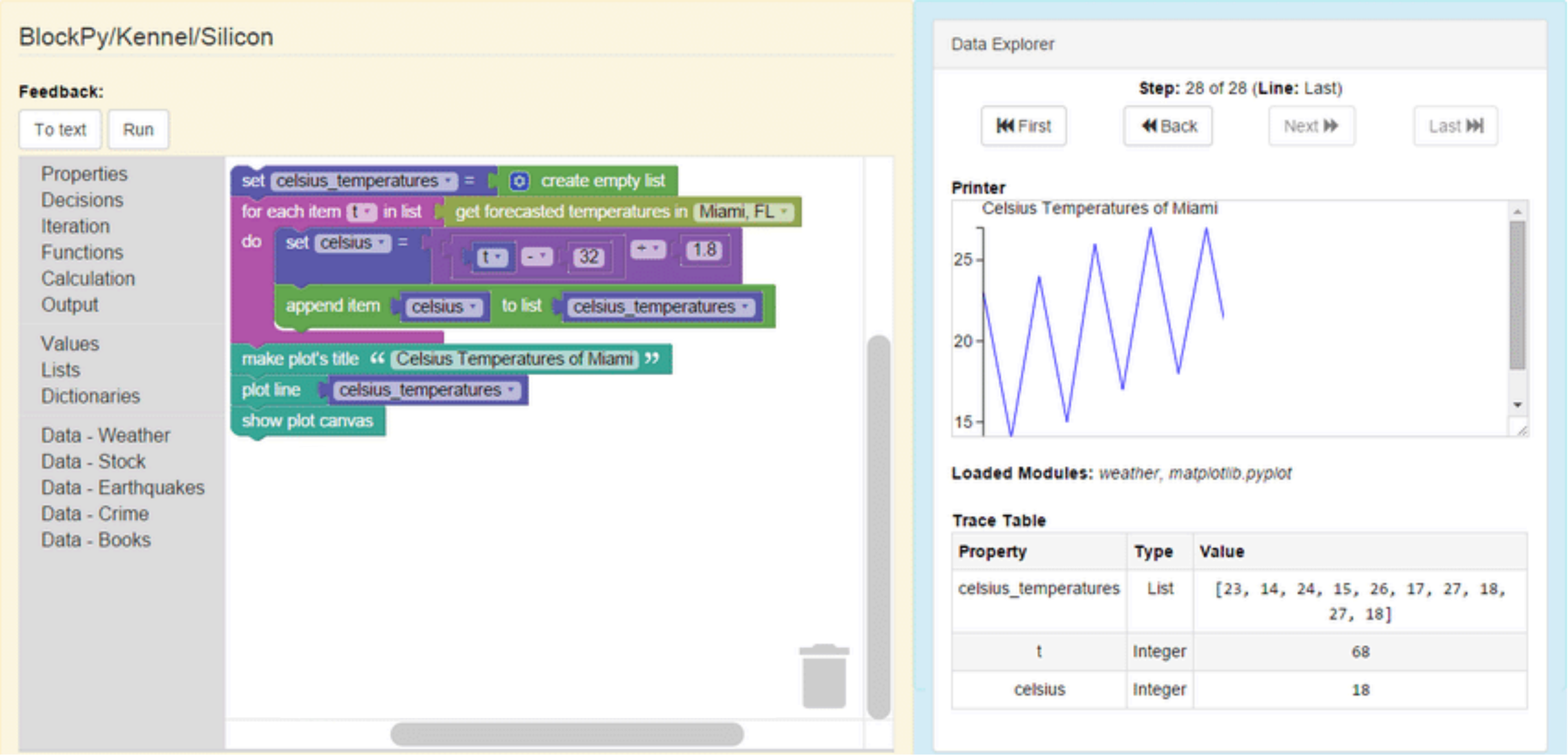
Basically, it's a matter of what editor we're using to build up programs in the language.

Python

```
import weather
import matplotlib.pyplot as plt


celsius_temperatures = []
for t in weather.get_forecasts("Miami, FL"):
    celsius = (t - 32) / 1.8
    celsius_temperatures.append(celsius)
plt.title("Celsius Temperatures of Miami")
plt.plot(celsius_temperatures)
plt.show()
```

…also Python


BlockPy/Kennel/Silicon
Feedback:
To text
Run
Properties
Decisions
Iteration
Functions
Calculation
Output
Values
Lists
Dictionaries
Data - Weather
Data - Stock
Data - Earthquakes
Data - Crime
Data - Books
set celsius_temperatures = create empty list
for each item t in list get forecasted temperatures in Miami, FL
do set celsius = t - 32 ÷ 1.8
append item celsius to list celsius_temperatures
make plot's title “Celsius Temperatures of Miami”
plot line celsius_temperatures
show plot canvas
Data Explorer
Step: 28 of 28 (Line: Last)
First
Back
Next
Last
Printer
Celsius Temperatures of Miami
25
20
15
Loaded Modules: weather, matplotlib.pyplot
Trace Table
Property
Type
Value
celsius_temperatures
List
[23, 14, 24, 15, 26, 17, 27, 18, 27, 18]
t
Integer
68
celsius
Integer
18

Programming Language vs. Programming Environment

Both of those were Python—same language.

One editor was clearly textual, and one editor was clearly visual.

One editor was (probably) non-projectional, and one editor was clearly projectional.

Programming Language vs. Programming Environment

Programming Language: For our purposes today, a code generator that takes ASTs as input

Programming Environment: The tool or tools we use for building up those ASTs

Programming Language vs. Programming Environment

Why do people get this confused?

Probably just because there are some *visual languages* that have only one interpreter, their own custom visual editor. If no one has written a parser for a text-based version of a given language, a visual environment may be the only way to write programs in it.

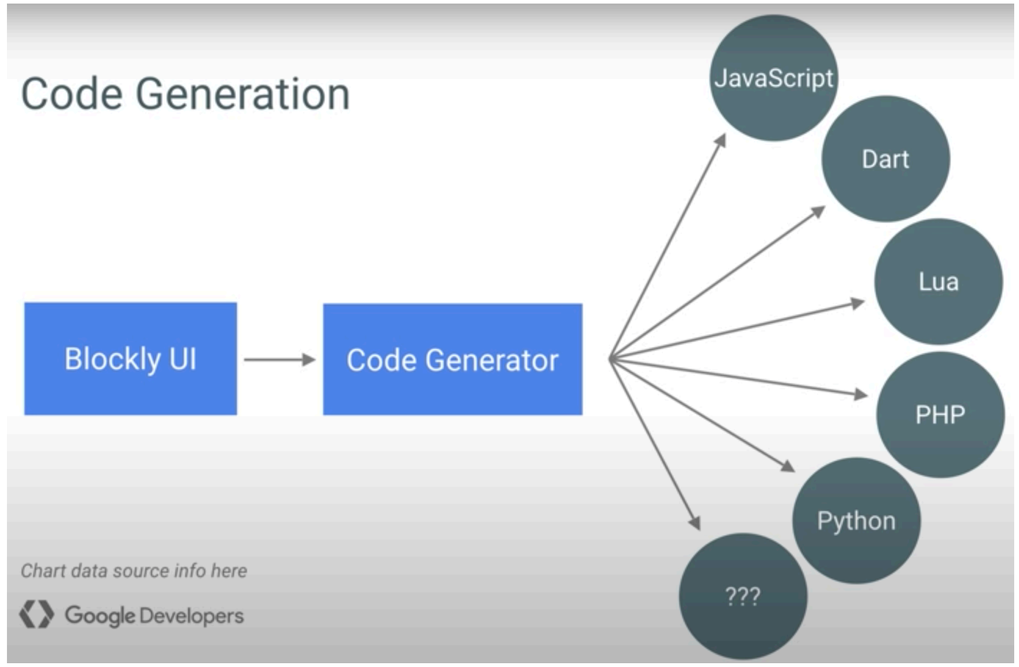
Programming Language vs. Programming Environment

Examples

Snap! : Both a programming language and a paired programming environment

Scratch : Same deal, both a programming language and a paired programming environment

Blockly : A library for making programming environments for whatever language you want

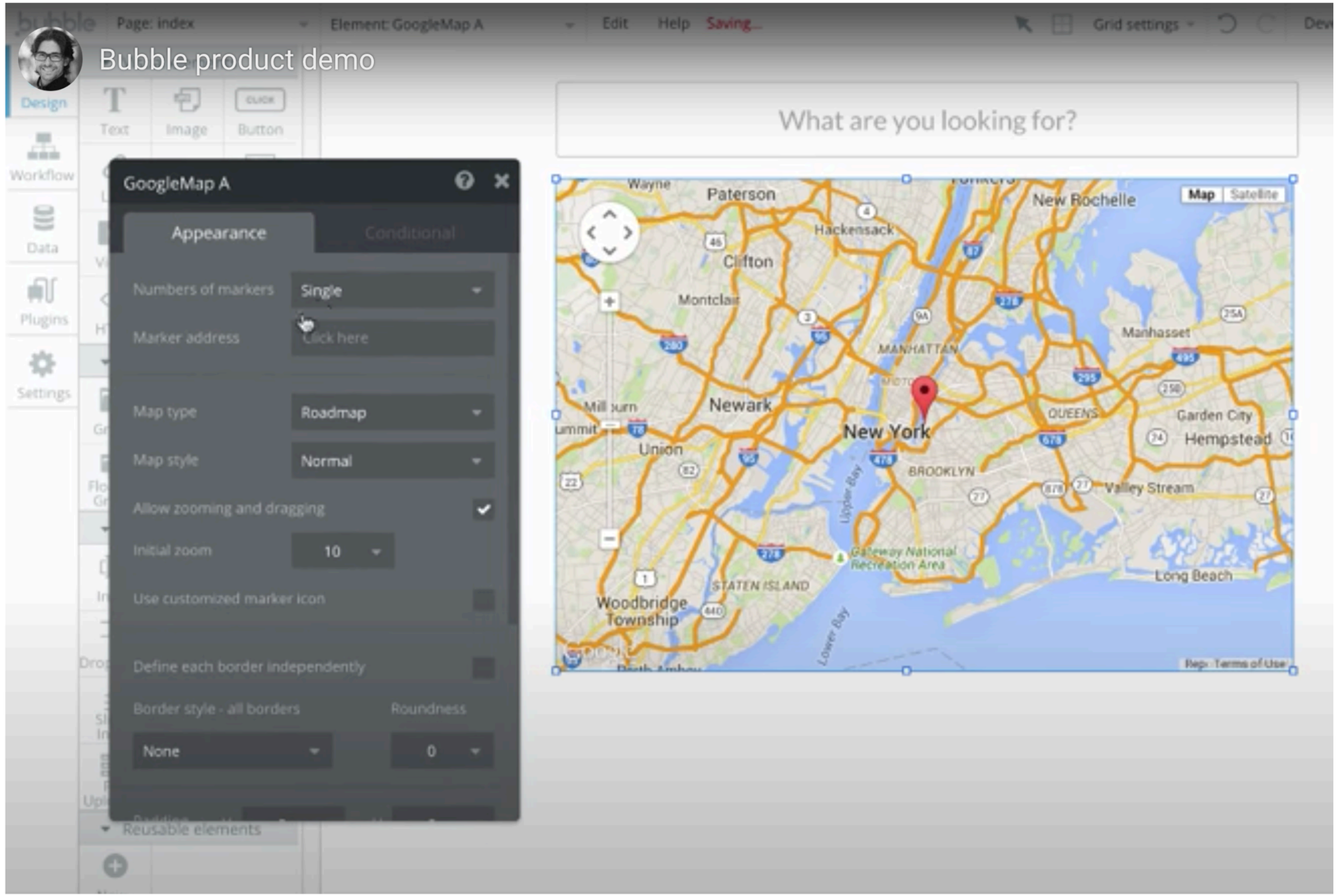


Projectional Editor vs. Visual Editor

Projectional Editor: Any editor (can be textual or visual) in which we build up programs by interacting directly with ASTs

Visual Editor: Any editor (can be projectional or non-projectional) in which we build programs by any means other than typing text in a textbox

Visual but not projectional



https://bubble.io/

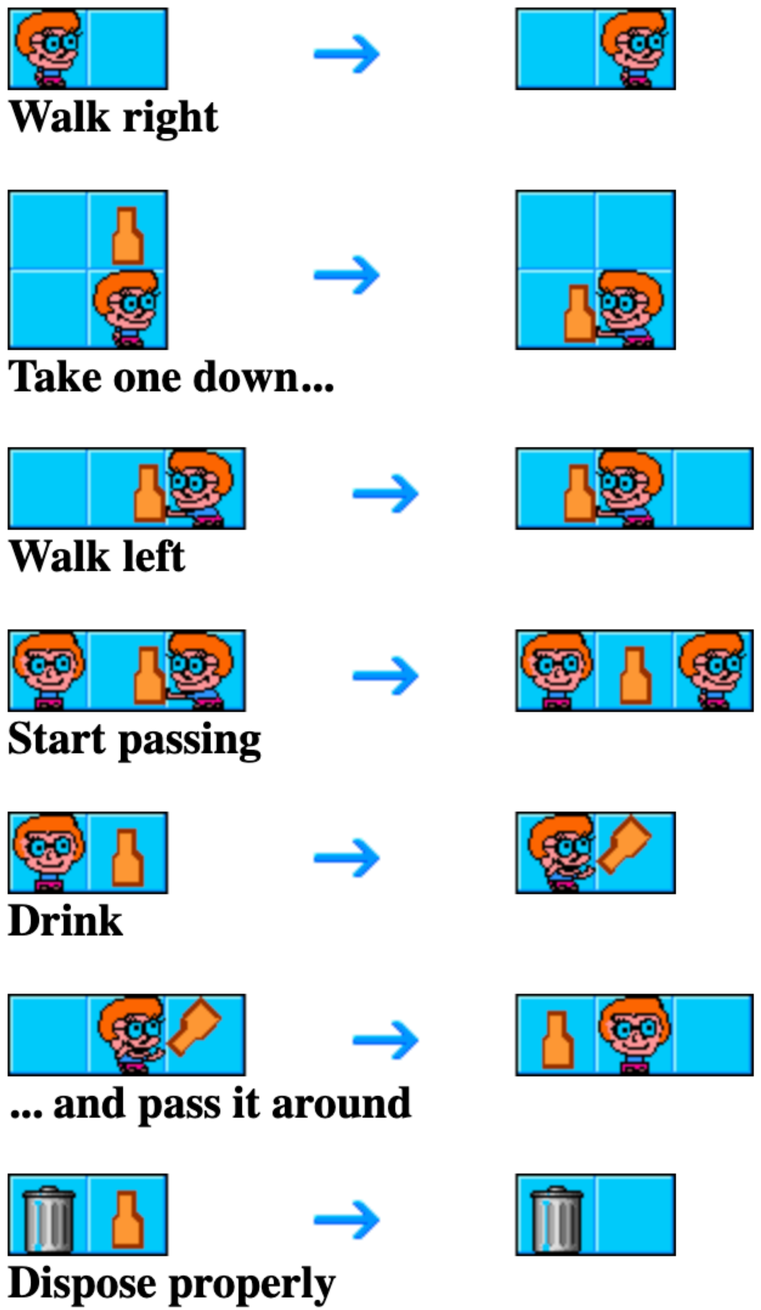
build and run web applications without code

Visual but not projectional

Stagecast Creator™

allows adults and children as young as 8 to build their own simulations and games

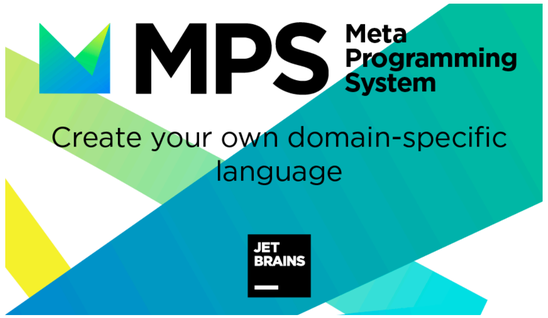
Here are all the rules in Creator for "99 Bottles of Root Beer":



Non-Projectional

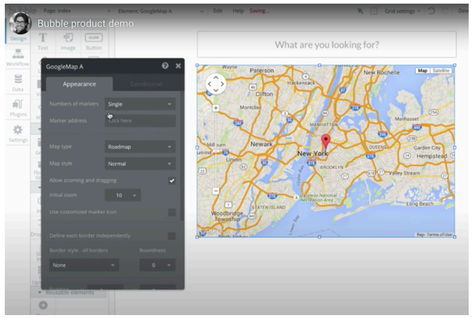
Projectional

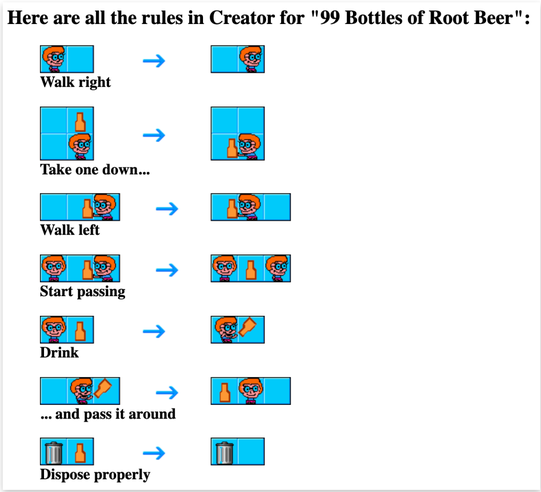
Textual

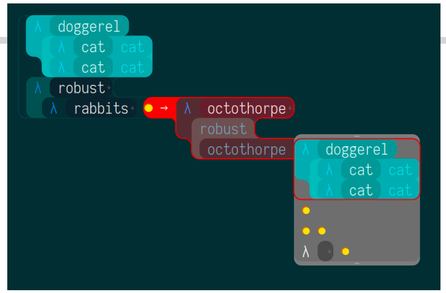


+
paredit

Visual







syntax errors vs. logical errors

Snap! Activity

https://snap.berkeley.edu/snap/snap.html

https://snap.berkeley.edu/snap/help/SnapManual.pdf

Before we switch to activity time…

Start thinking about final projects.
More context on Thursday!

Snap! Activity

We're about to build some small extensions to a language that has a single interpreter that lives in a projectional editor.

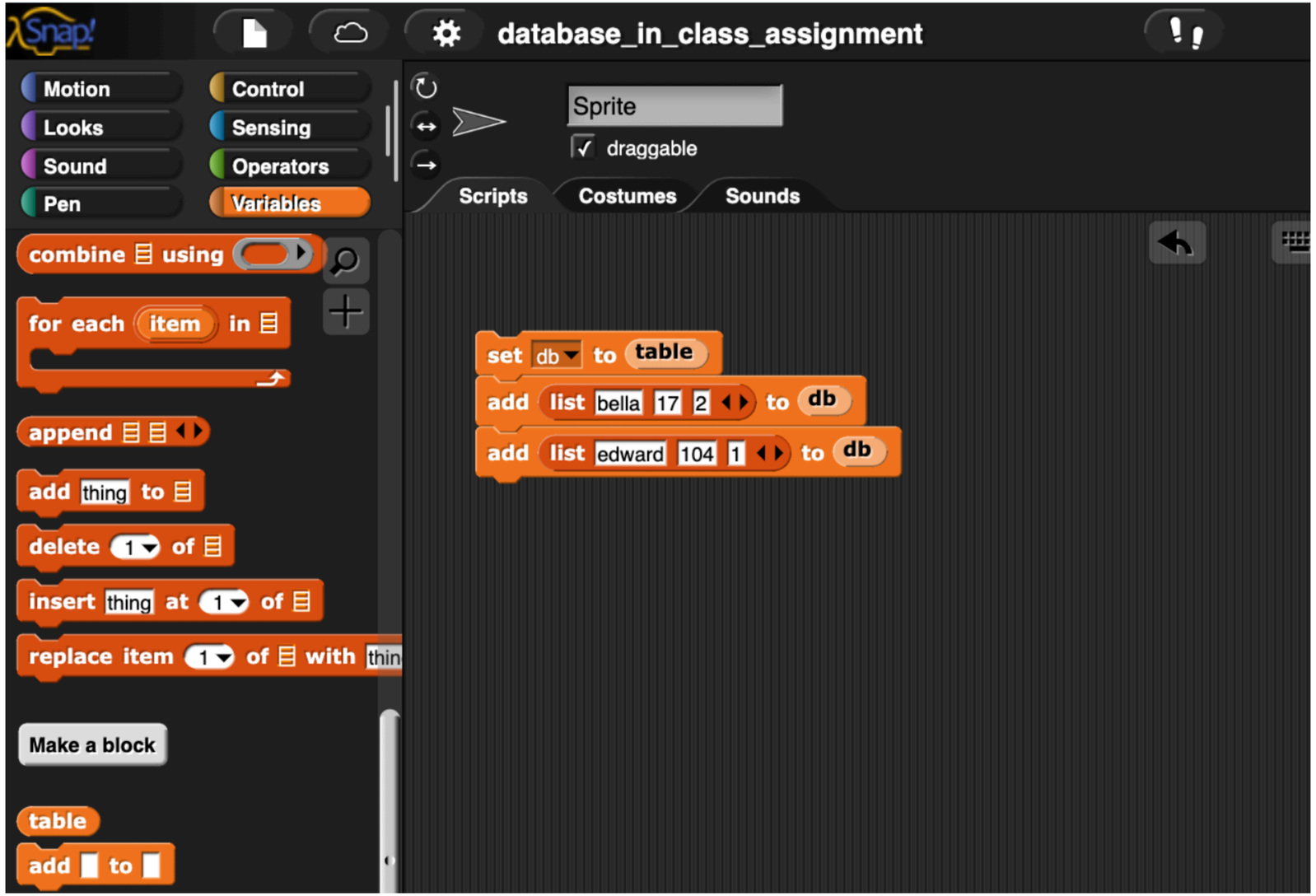
Notes:

- This isn't usually how people implement languages (even for projectional editors).
- However! It gives us a 2 birds 1 stone thing—we can experience *using* a projectional editor and building abstractions for use in a projectional editor at the same time!
- Intentionally slightly less directed than our usual activities, in hopes y'all will explore the Snap! landscape a bit.

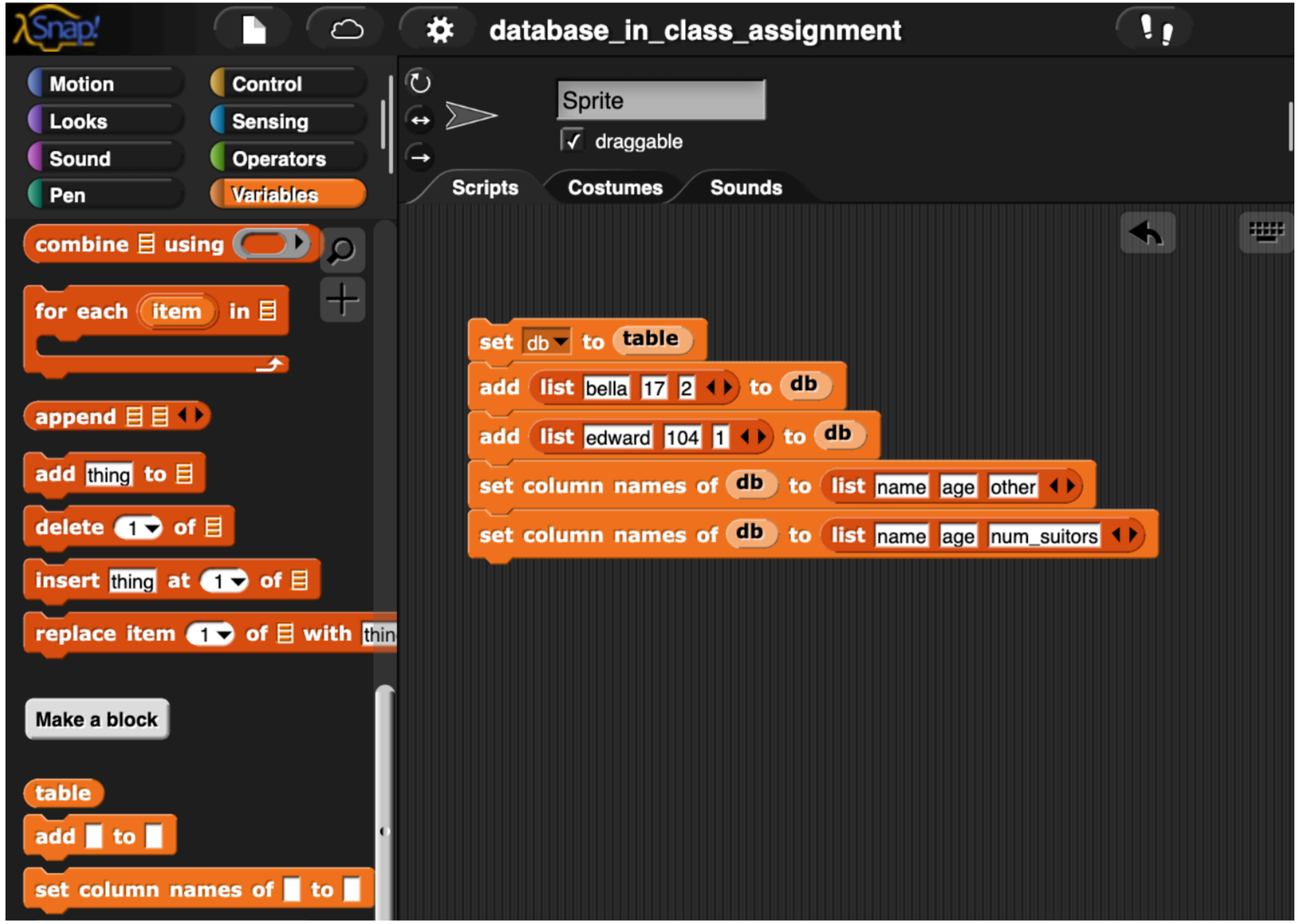
Snap! Activity

- One tip before we get started.
  - This seems like it shouldn’t matter, but it can get annoying, and no one ever figures it out themselves… If you end up with a “variable watcher” in the “stage” (white box in upper right) that you don’t want to show anymore, and you can’t get rid of it, drag it to the toolbox on the left that shows all the available blocks.

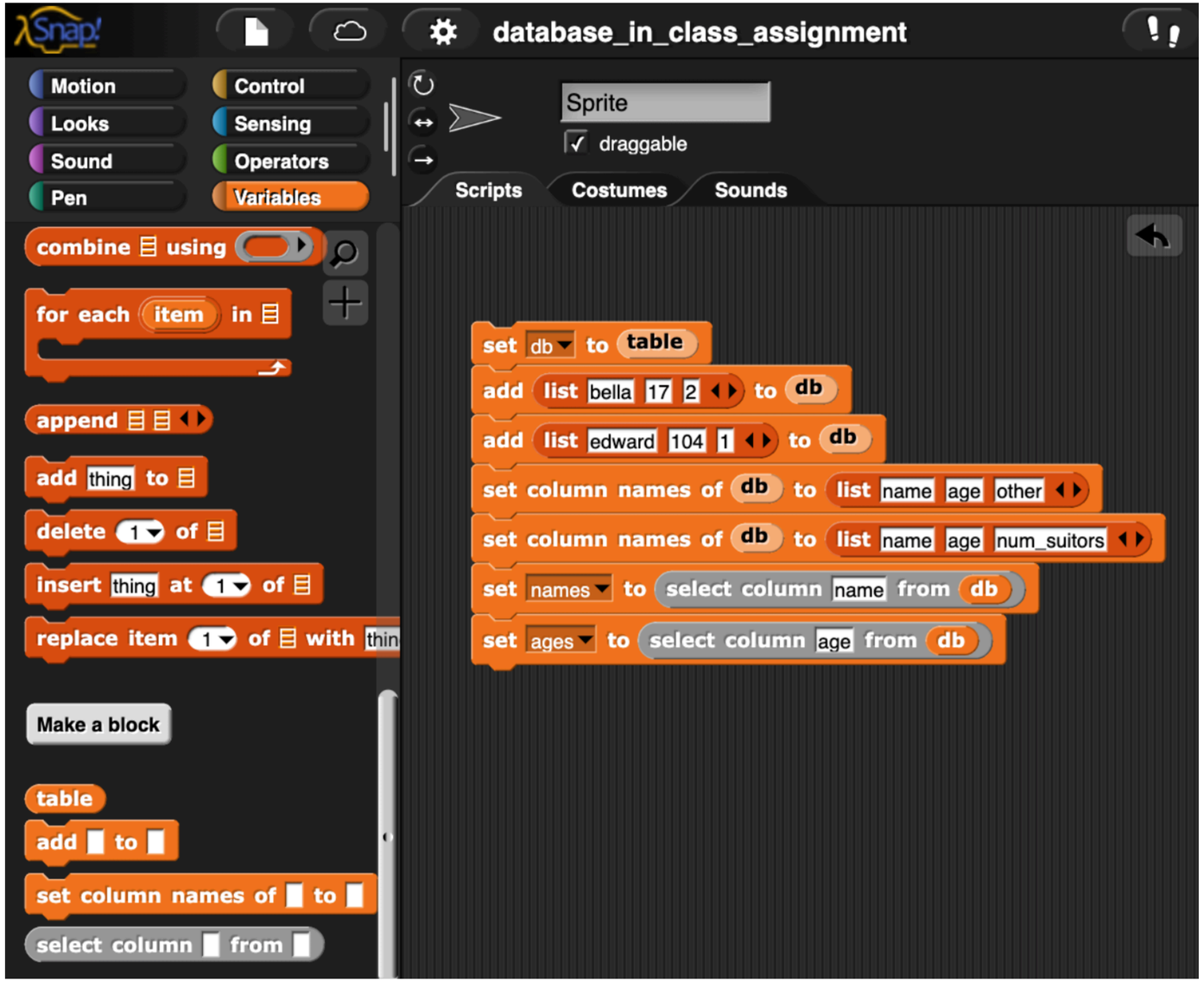
Snap! Activity - Stage 1



Snap! Activity - Stage 2



Snap! Activity - Stage 3



Snap! Activity - Stage 4

HW Assignment 6
Note: Doesn’t have to be in Snap!
(And don’t worry, just because we’re starting HW already doesn’t mean we’re done with structure editors!)