

Structure/Structured/ Projectional Editors

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 readings for today, come up with:
 - 3 task-audience combinations for which you'd instantiate a language in a non-projectional editor
 - 3 task-audience combinations for which you'd instantiate a language in a projectional editor

structure editor

==

structured editor

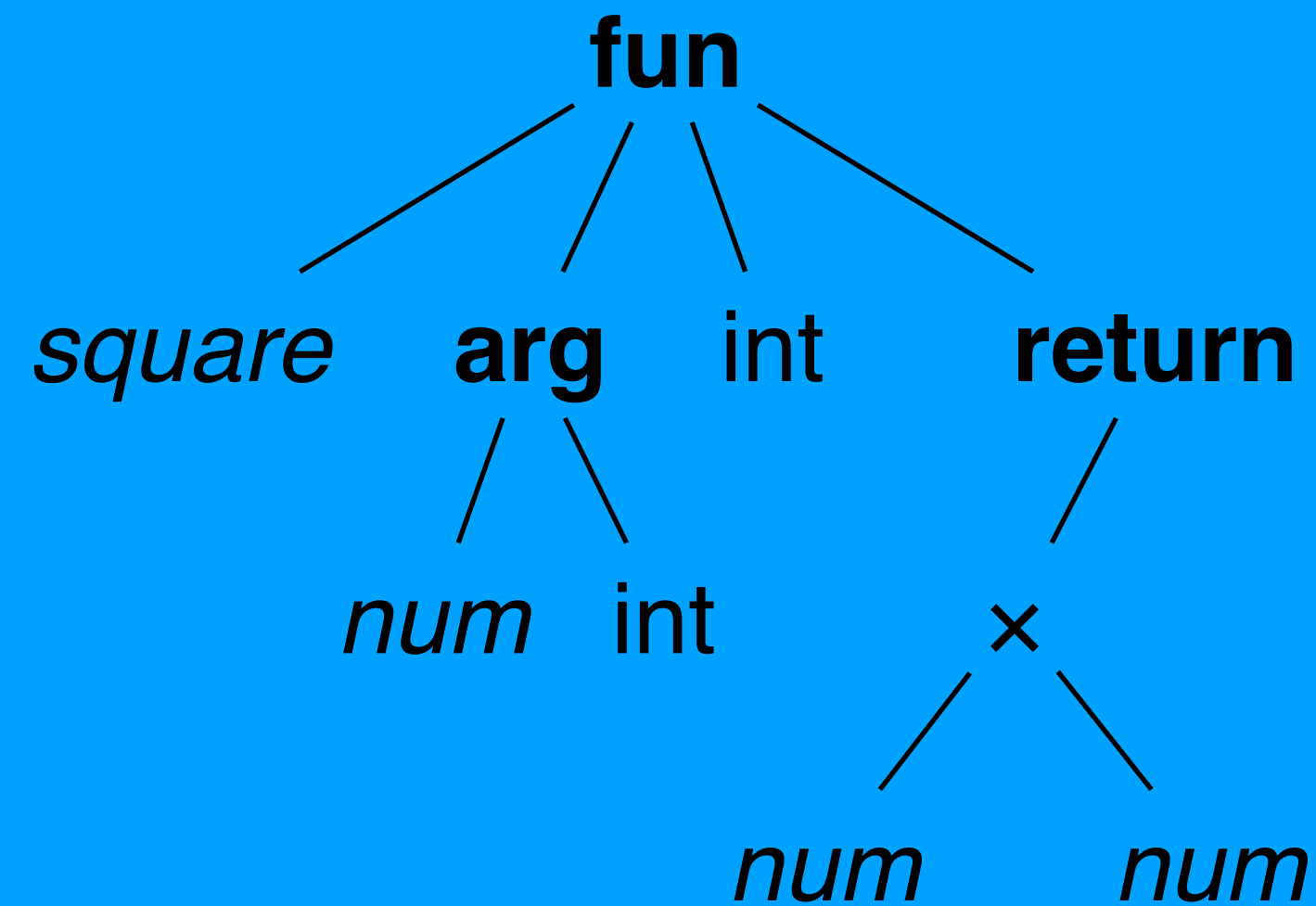
==

projectional editor

What's happening inside my compiler?

parser

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



Compiler

code generator

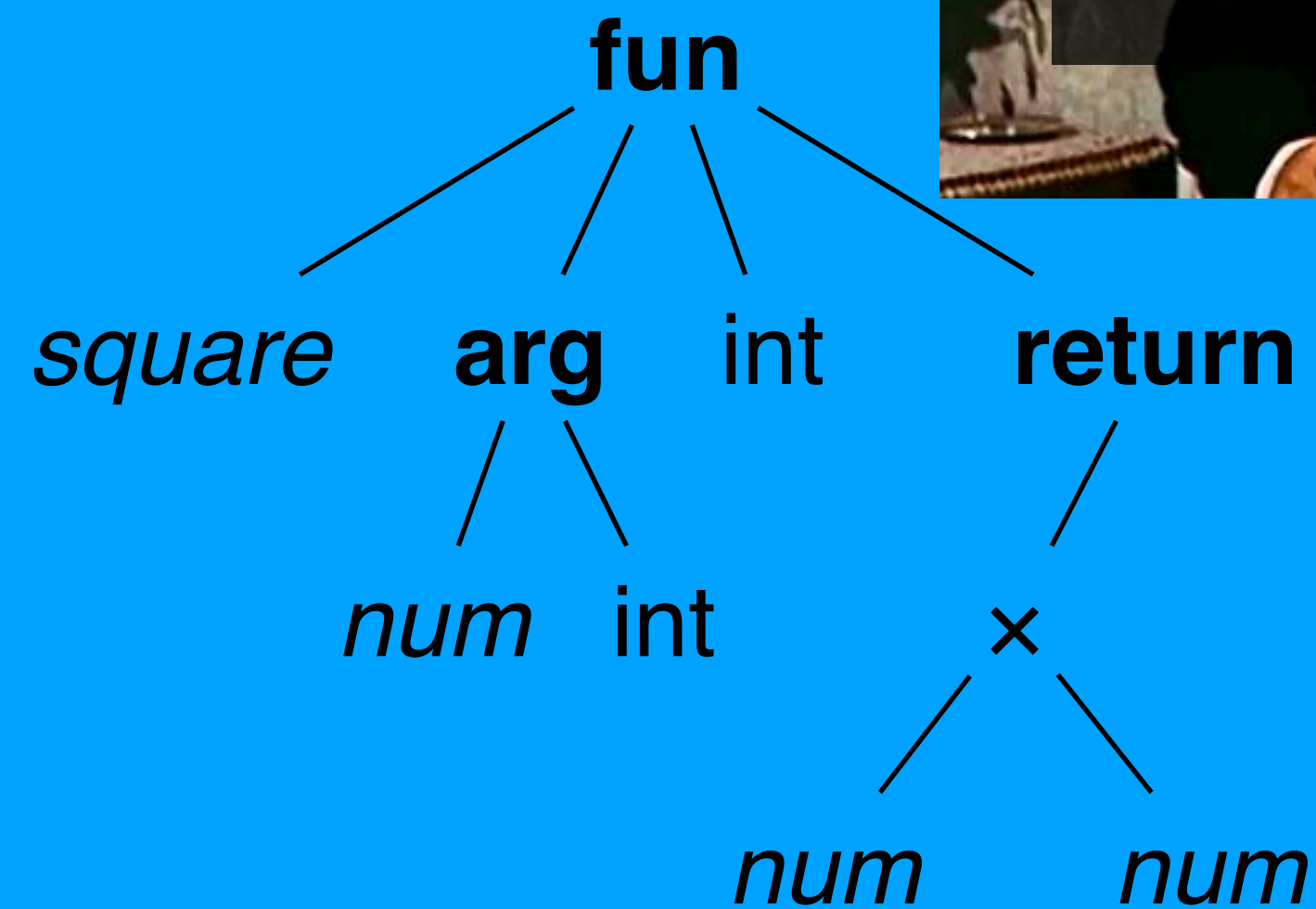
```
1 square(int):
2     push    rbp
3     mov     rbp, rsp
4     mov     DWORD PTR [rbp-4], edi
5     mov     eax, DWORD PTR [rbp-4]
6     imul    eax, eax
7     pop     rbp
8     ret
```

Assembly Language

What's happening inside my compiler

parser

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



Compiler



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 compiler?

parser

AST

fun

square **arg** *int* **return**

num *int* × *num* *num*

code generator

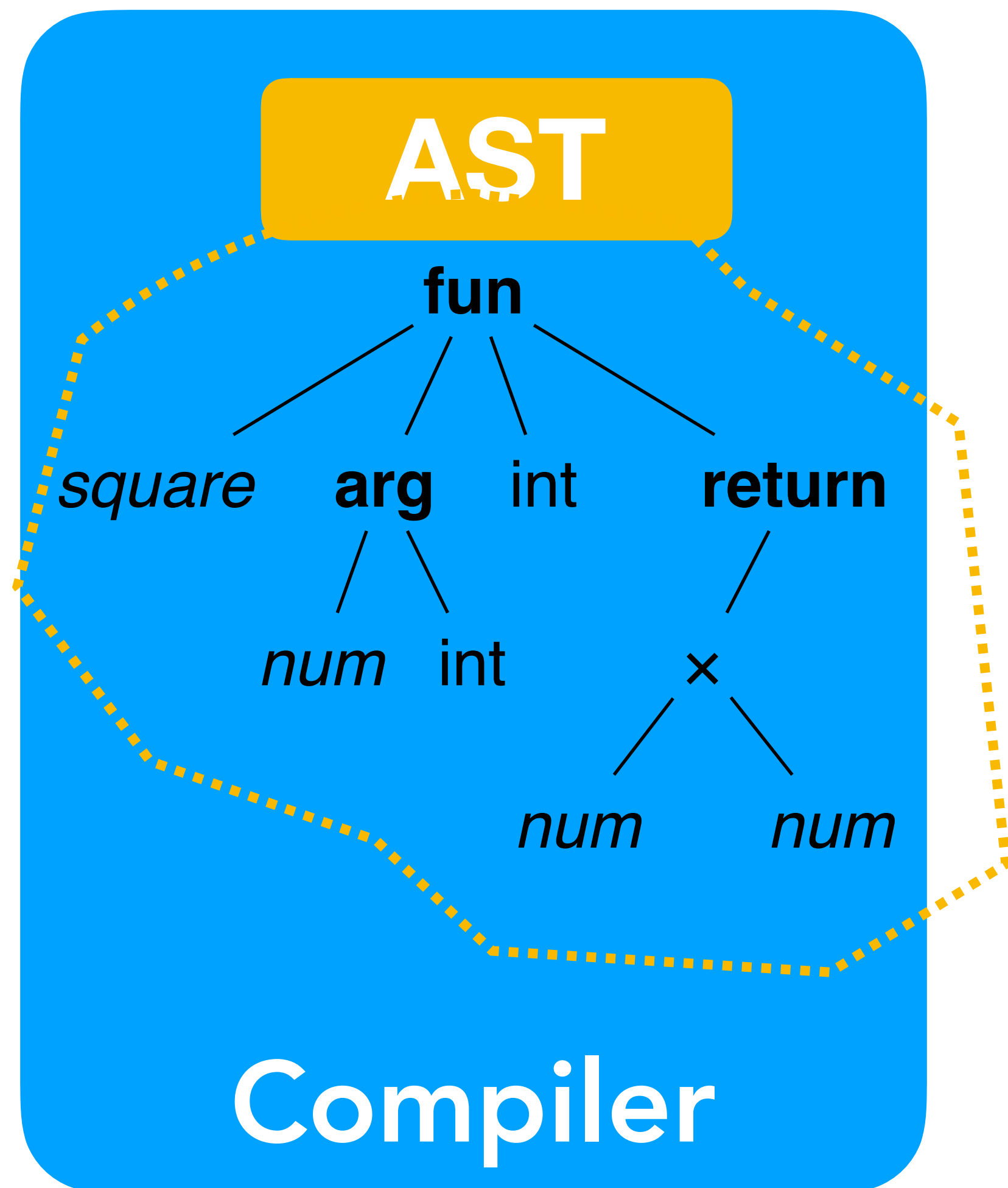
```
1 square(int):  
2     push    rbp  
3     mov     rbp, rsp  
4     mov     DWORD PTR [rbp-4], edi  
5     mov     eax, DWORD PTR [rbp-4]  
6     imul    eax, eax  
7     pop     rbp  
8     ret
```

Assembly Language

Compiler

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


Abstract Syntax Tree (AST)

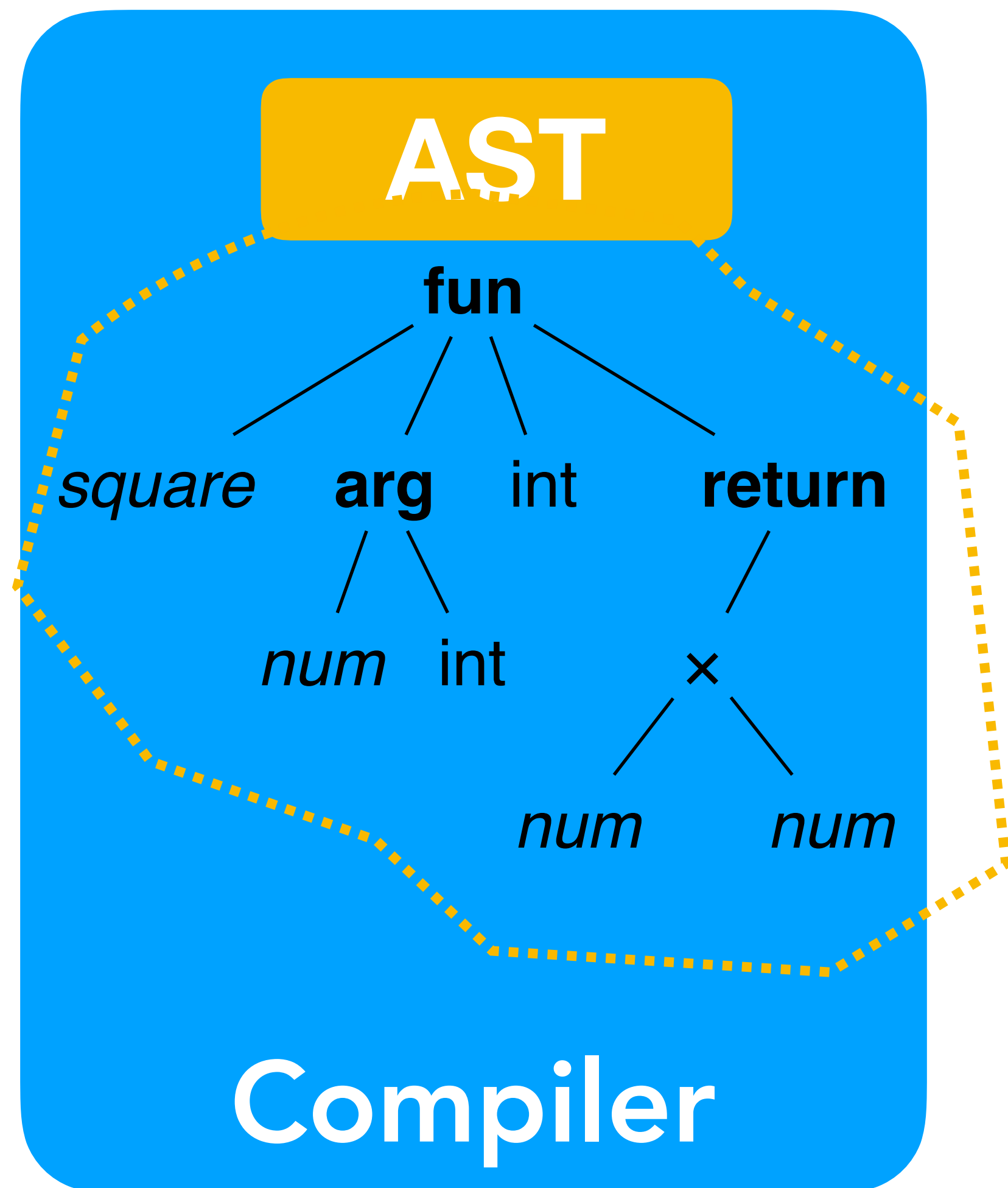


Abstract because we're not putting in every detail of the actual programming language syntax. (E.g., we've dropped all those pesky semicolons and parens.)

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

```
1 import weather
2 import matplotlib.pyplot as plt
3
4
5 celsius_temperatures = []
6 for t in weather.get_forecasts("Miami, FL"):
7     celsius = (t - 32) / 1.8
8     celsius_temperatures.append(celsius)
9 plt.title("Celsius Temperatures of Miami")
10 plt.plot(celsius_temperatures)
11 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



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.

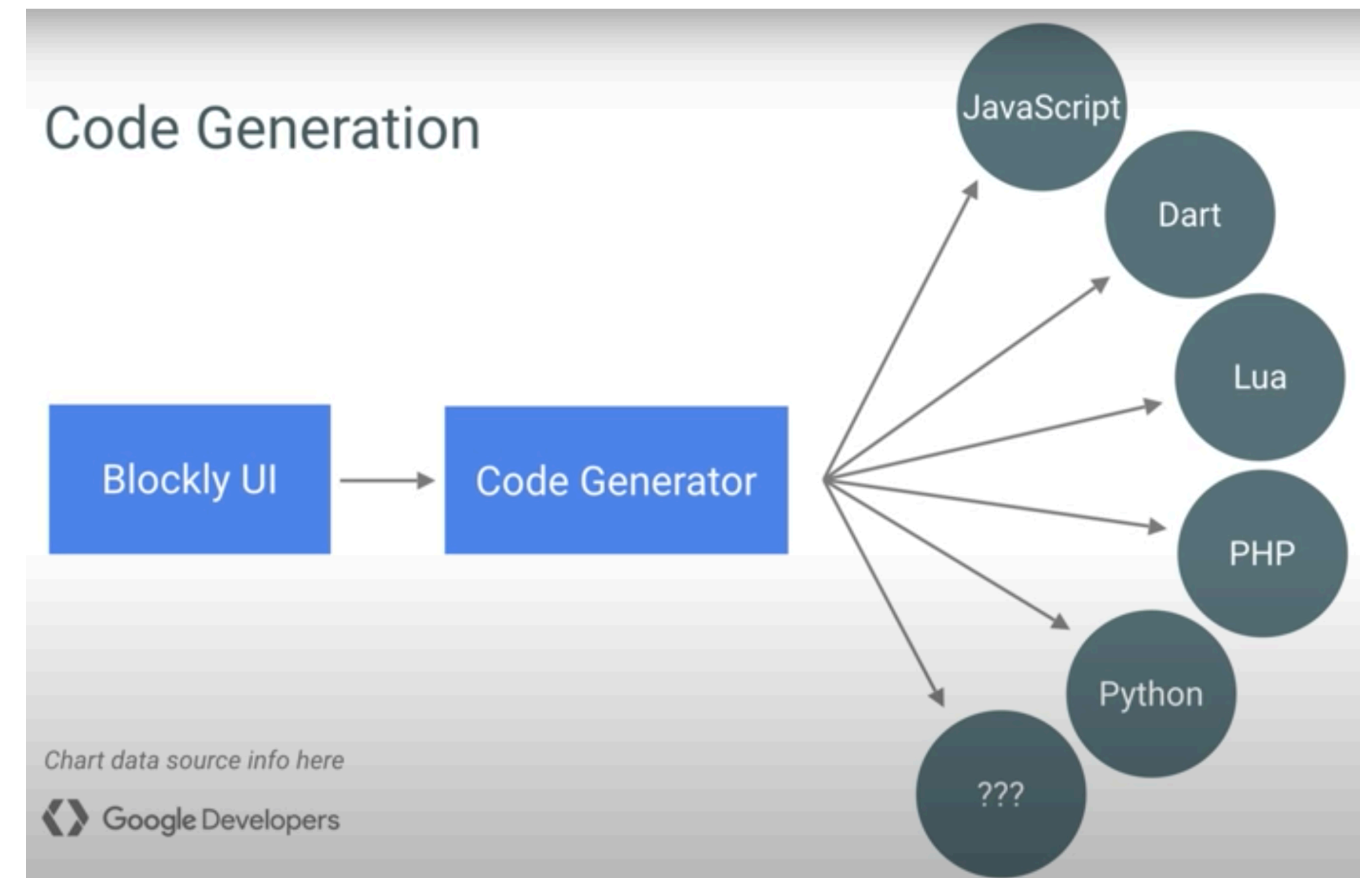
Programming Language vs. Programming Environment

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

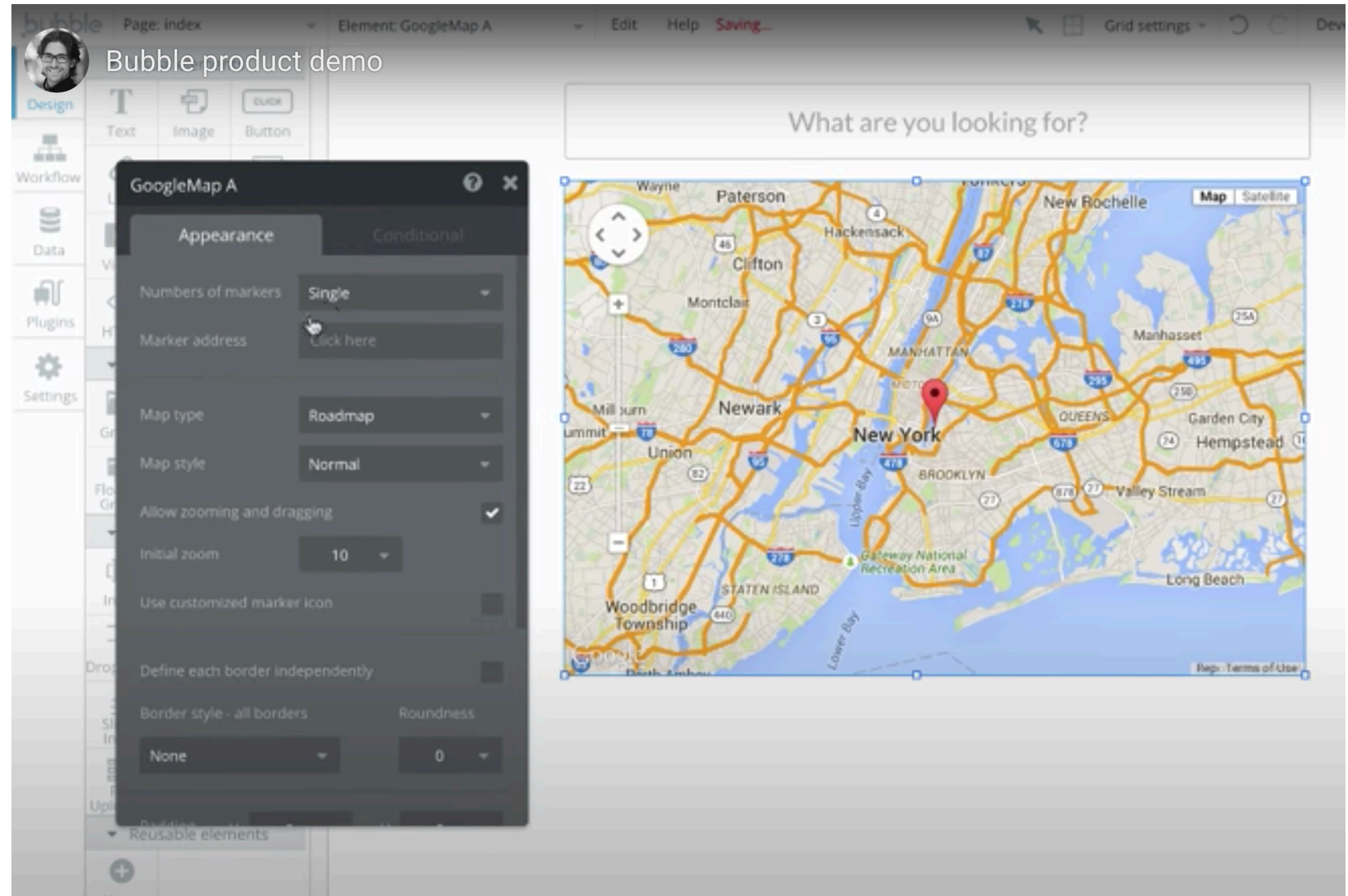
Projectional Editor: Any editor (can be textual or visual) in which we build up programs by interacting directly with ASTs

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":



Walk right



Take one down...



Walk left



Start passing



Drink



... and pass it around



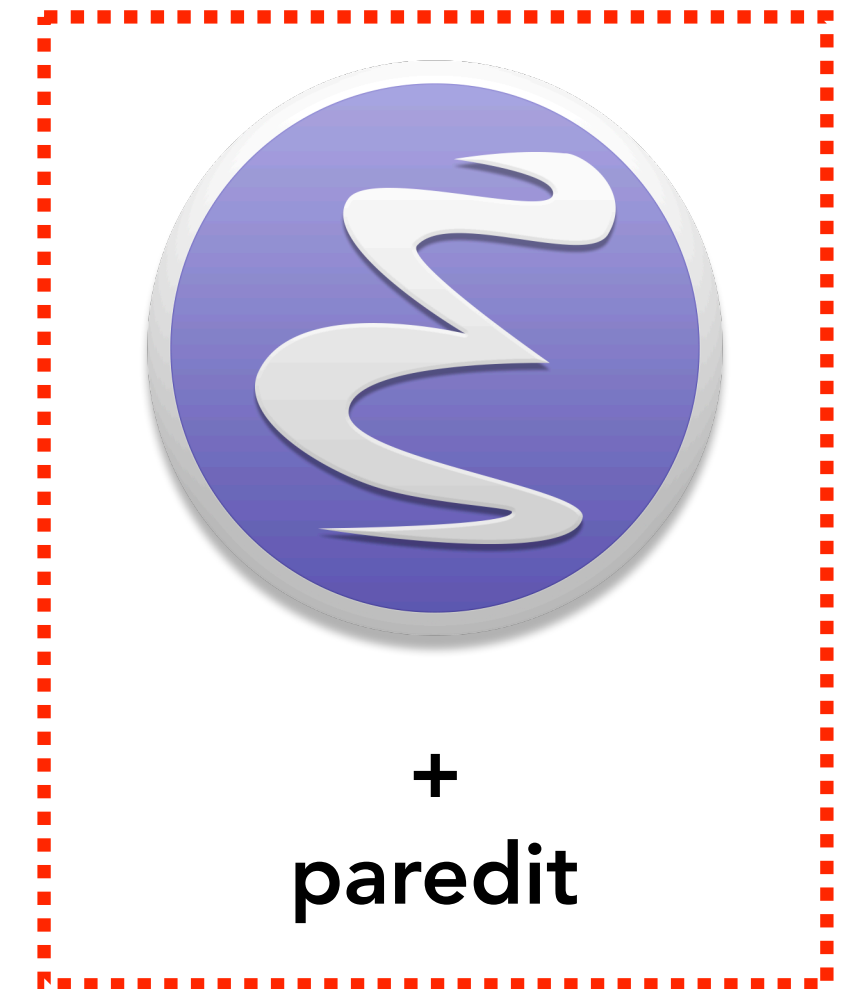
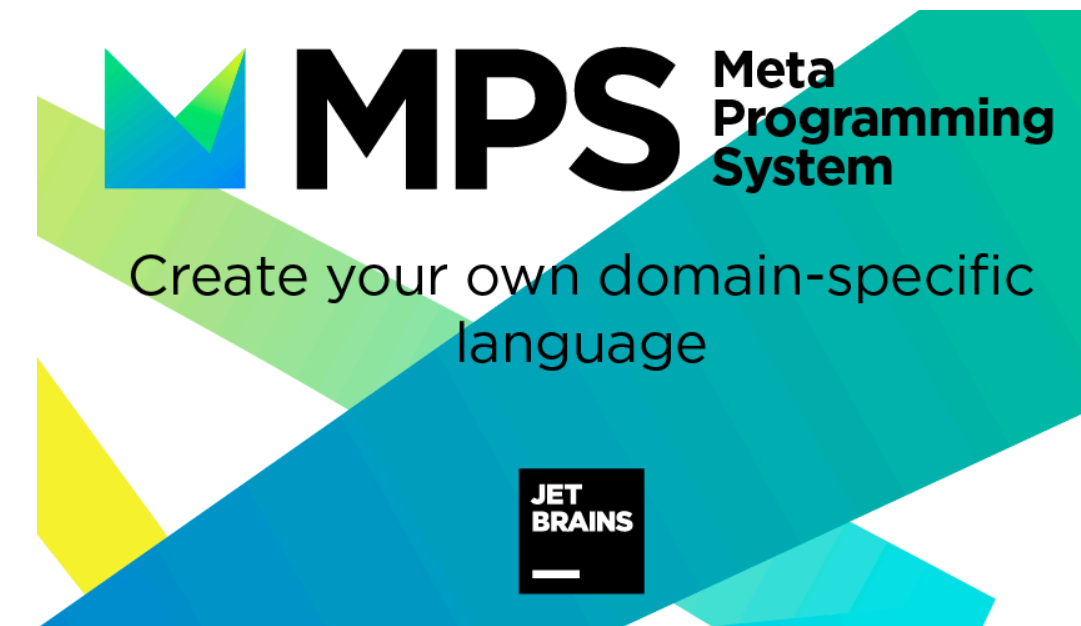
Dispose properly

Textual

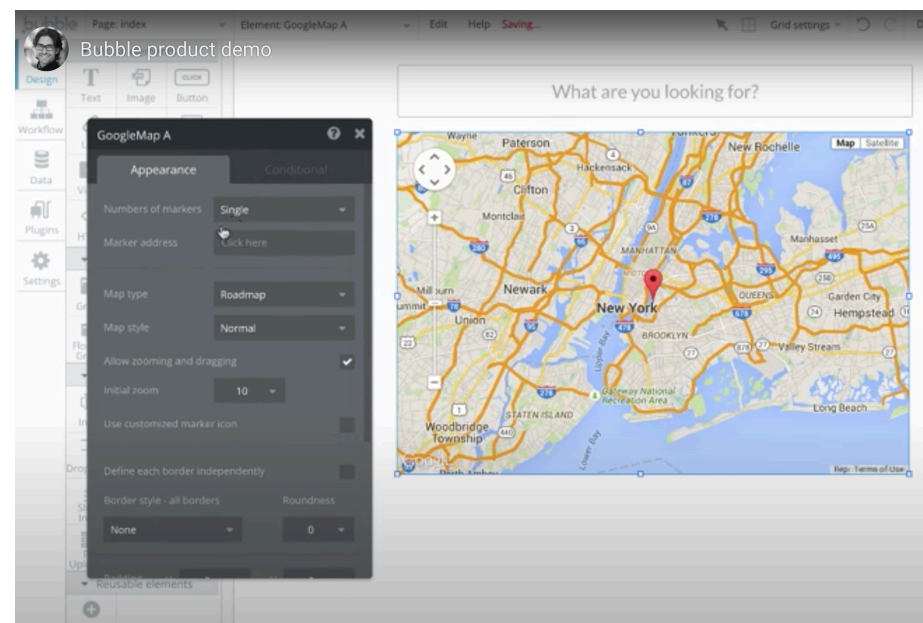
Non-Projectional



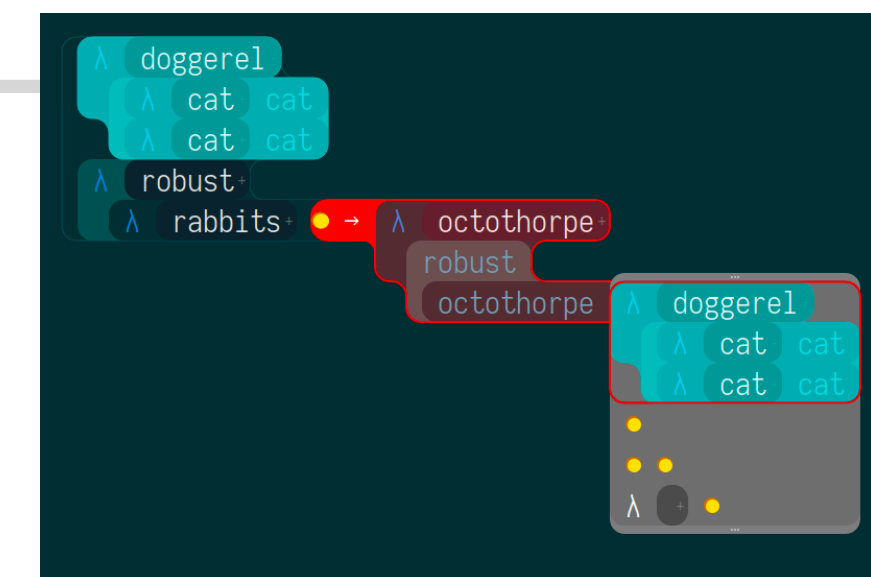
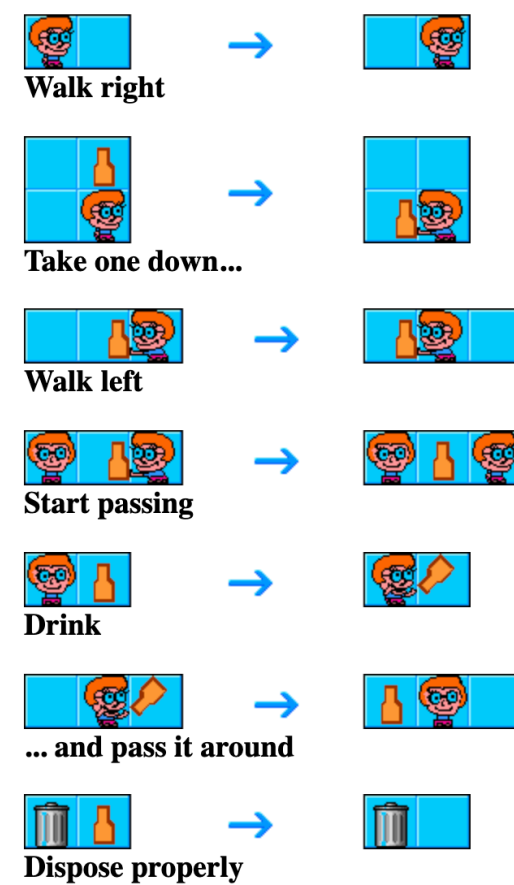
Projectional



Visual



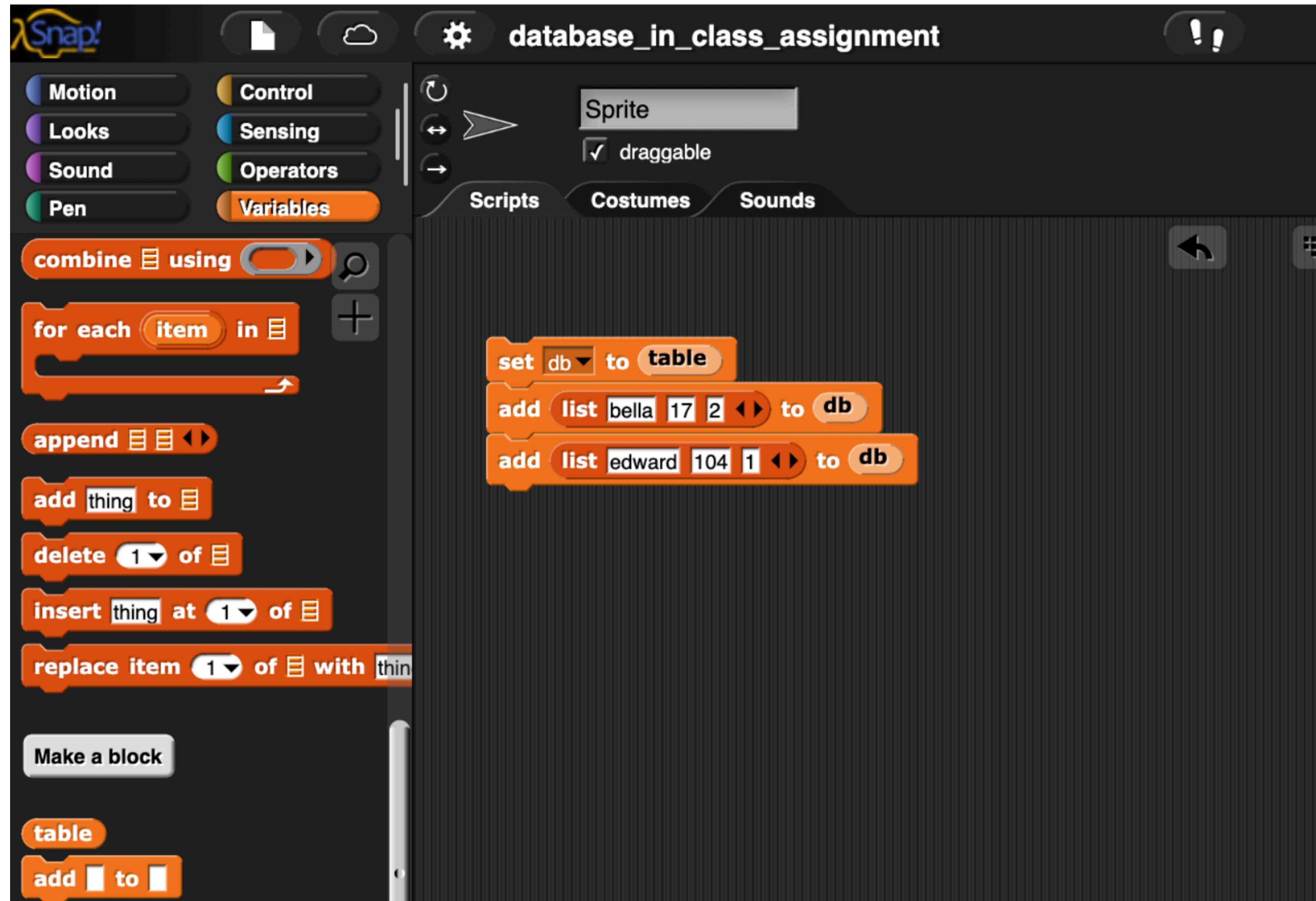
Here are all the rules in Creator for "99 Bottles of Root Beer":



Snap! Activity

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

Snap! Activity - Stage 1



Snap! Activity - Stage 2

The screenshot shows the Snap! programming environment with a project titled "database_in_class_assignment". The interface includes a left sidebar with category tabs (Motion, Looks, Sound, Pen, Control, Sensing, Operators, Variables) and a right sidebar with tabs for Scripts, Costumes, and Sounds. The main workspace displays a script for a "Sprite" that is draggable. The script consists of the following blocks:

- set db to table
- add list bella 17 2 to db
- add list edward 104 1 to db
- set column names of db to list name age other
- set column names of db to list name age num_suitors

The left sidebar also shows a "combine using" block, a "for each item in" loop, and an "append" block. At the bottom of the left sidebar, there is a "Make a block" button and a "table" block with "add" and "set column names of" sub-blocks.

Snap! Activity - Stage 3

The screenshot shows the Snap! workspace titled "database_in_class_assignment". The left sidebar contains category tabs: Motion, Looks, Sound, Pen, Control, Sensing, Operators, and Variables. The main workspace shows a script for a "Sprite" object, which is draggable. The script consists of the following blocks:

- set db to table
- add list bella 17 2 to db
- add list edward 104 1 to db
- set column names of db to list name age other
- set column names of db to list name age num_suitors
- set names to select column name from db
- set ages to select column age from db

Below the script, there is a "Make a block" button and a table structure with the following blocks:

- table
- add to
- set column names of to
- select column from

names	
1	bella
2	edward
length: 2	

ages	
1	17
2	104
length: 2	

Snap! Activity - Stage 4

HW Assignment 6

<http://schasins.com/cs294-usable-programming-2020/assets/assignments/a6.pdf>

Note: Doesn't have to be in Snap!

Goal for next reading

- Prepare to write a program slicer! Understand the basics in preparation for writing your own.