

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

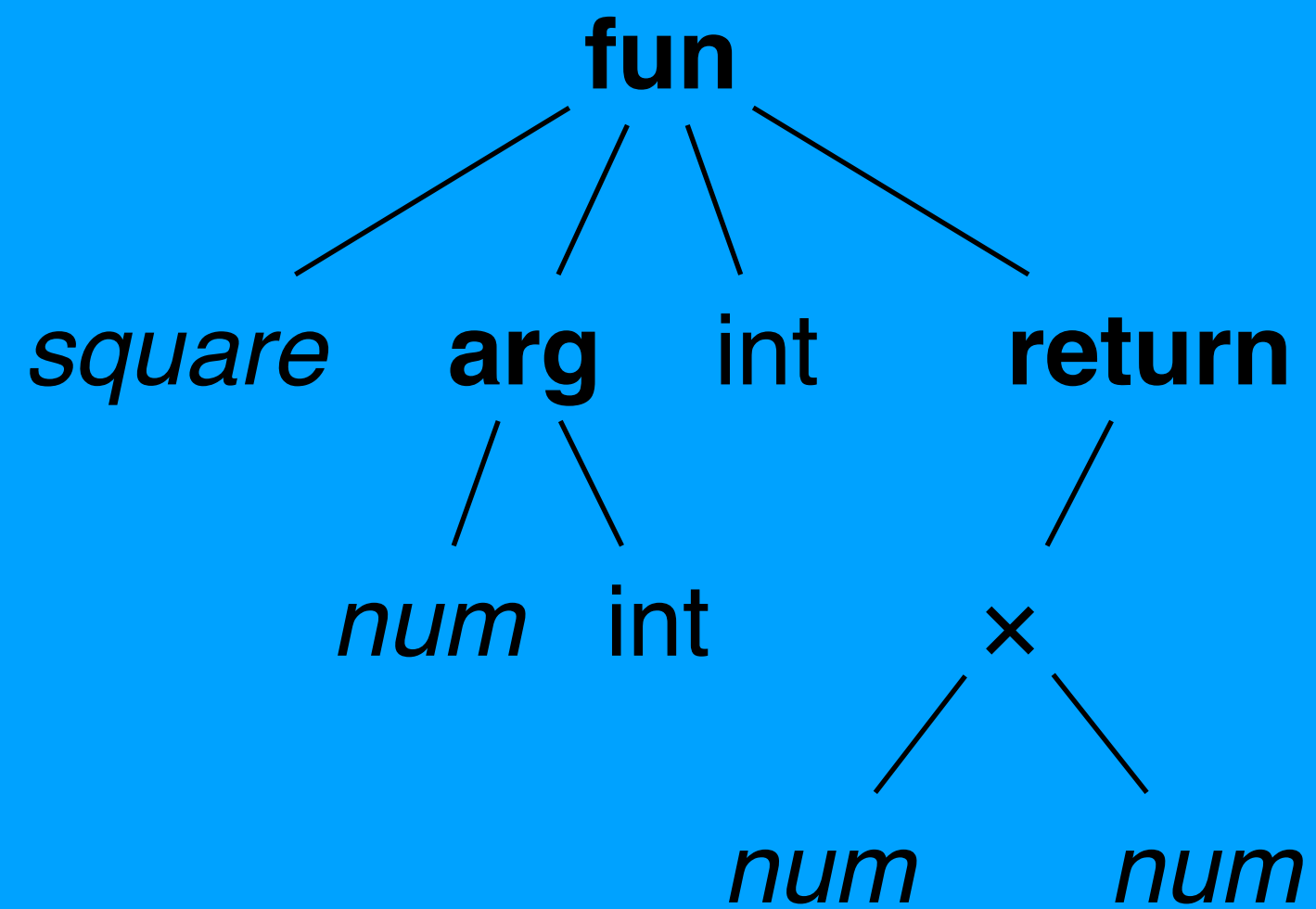
==

projectional editor

What's happening inside my compiler?

parser

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



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

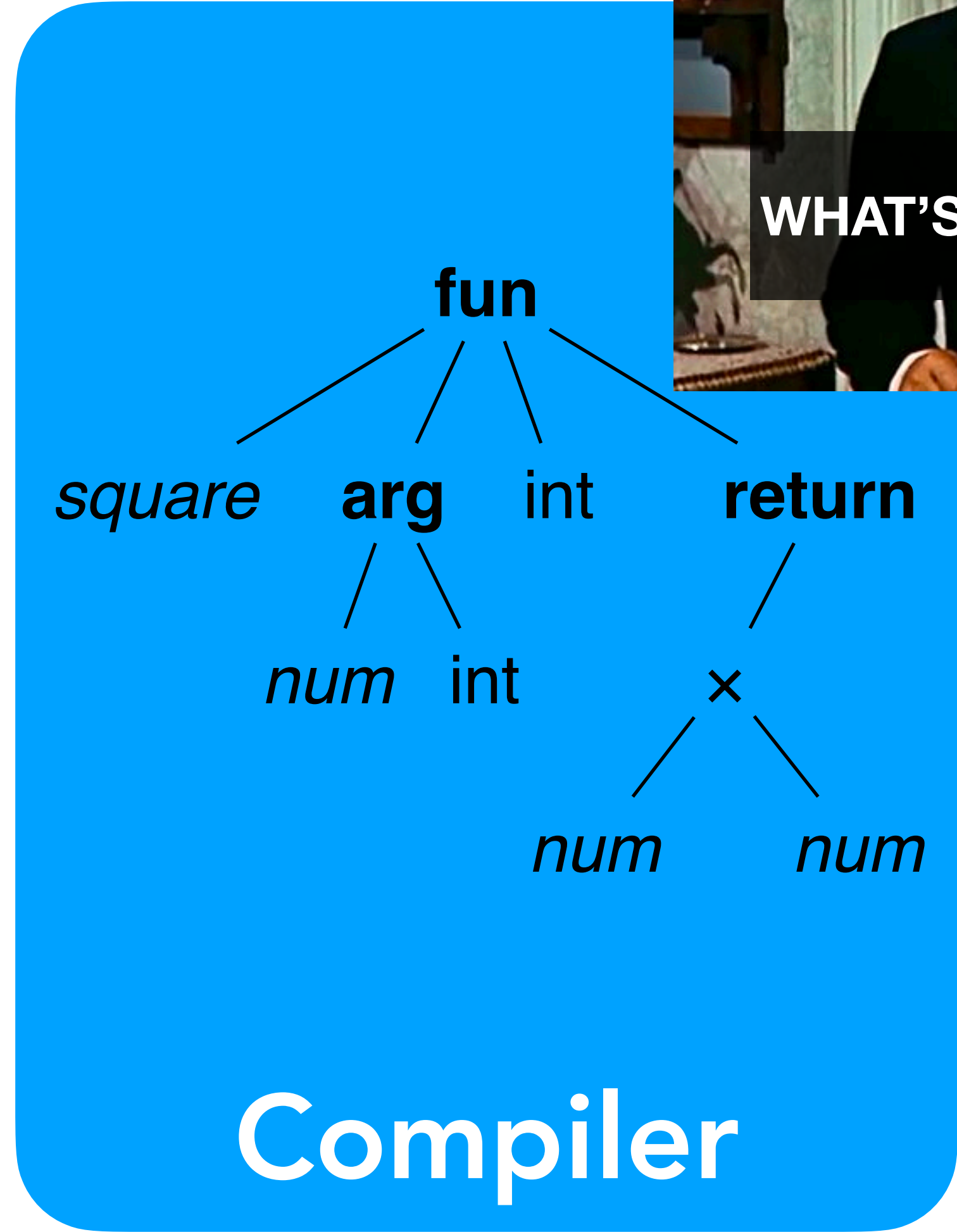
Compiler

What's happening inside my compiler



parser

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



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

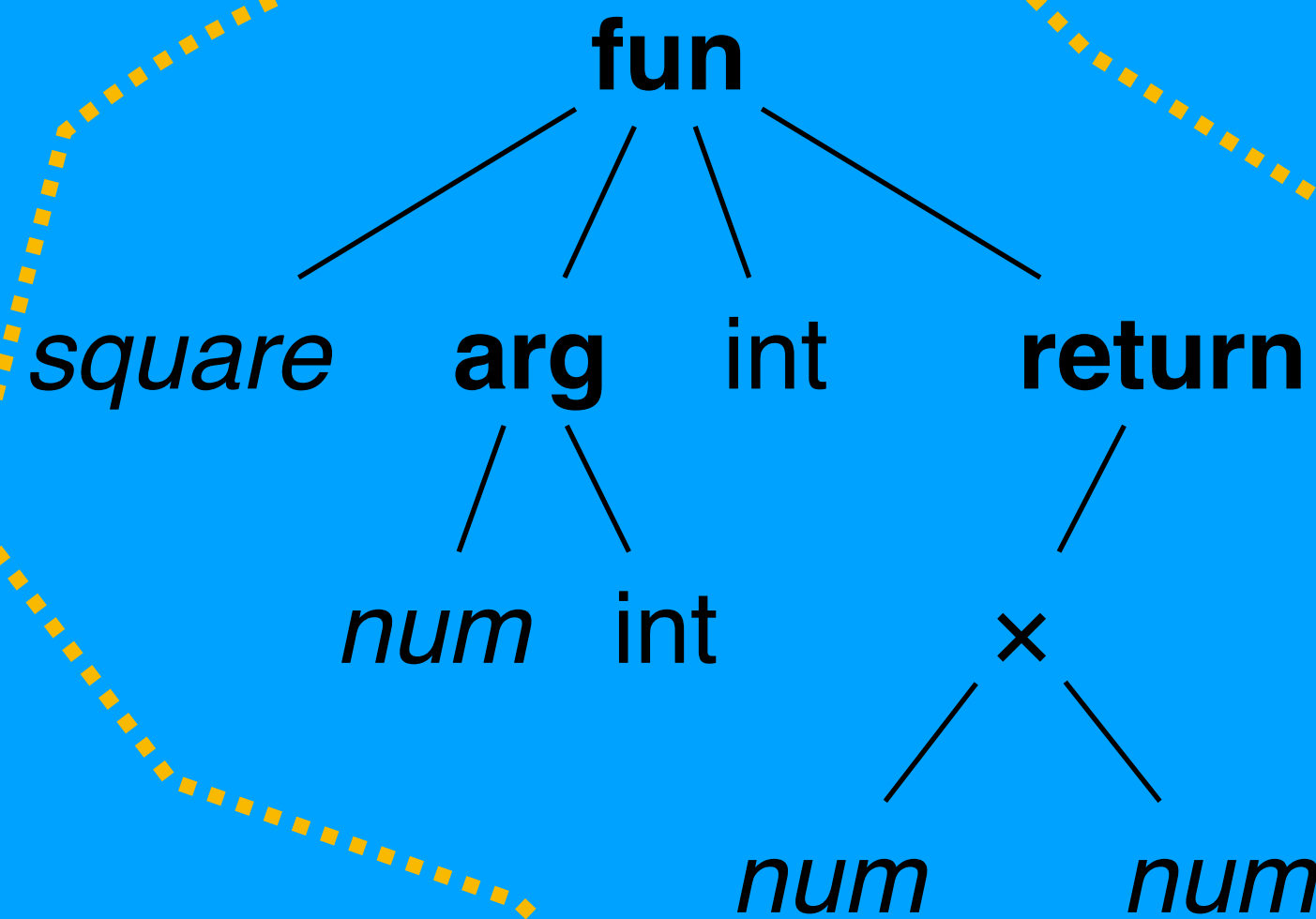
What's happening inside my compiler?

parser

AST

code generator

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

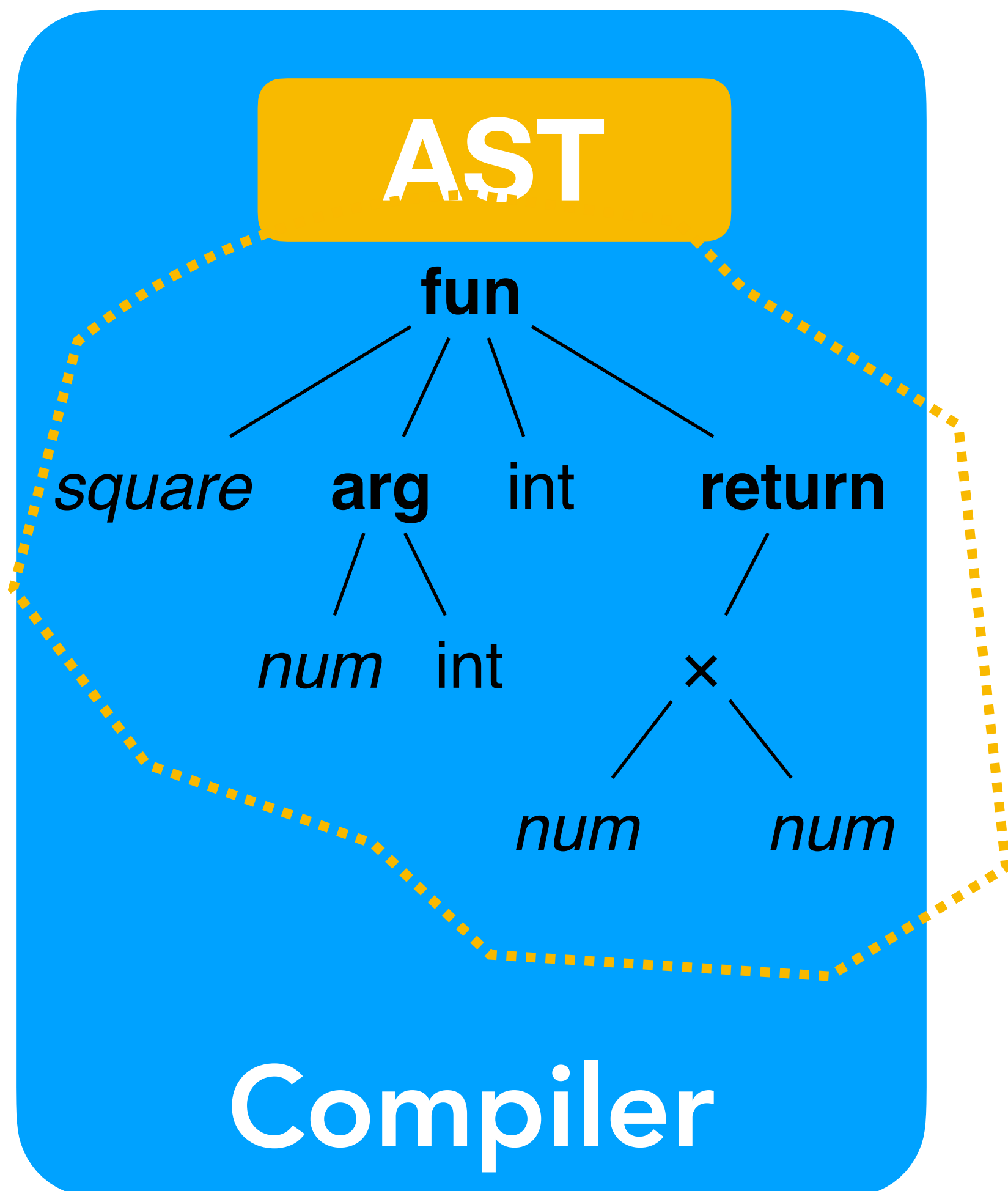


```
1 square(int):
2     push    rbp
3     mov     rbp, rsp
4     mov     DWORD PTR [rbp-4], edi
5     mov     eax, DWORD PTR [rbp-4]
6     imul   eax, eax
7     pop     rbp
8     ret
```

Assembly Language

Compiler

Abstract Syntax Tree (AST)

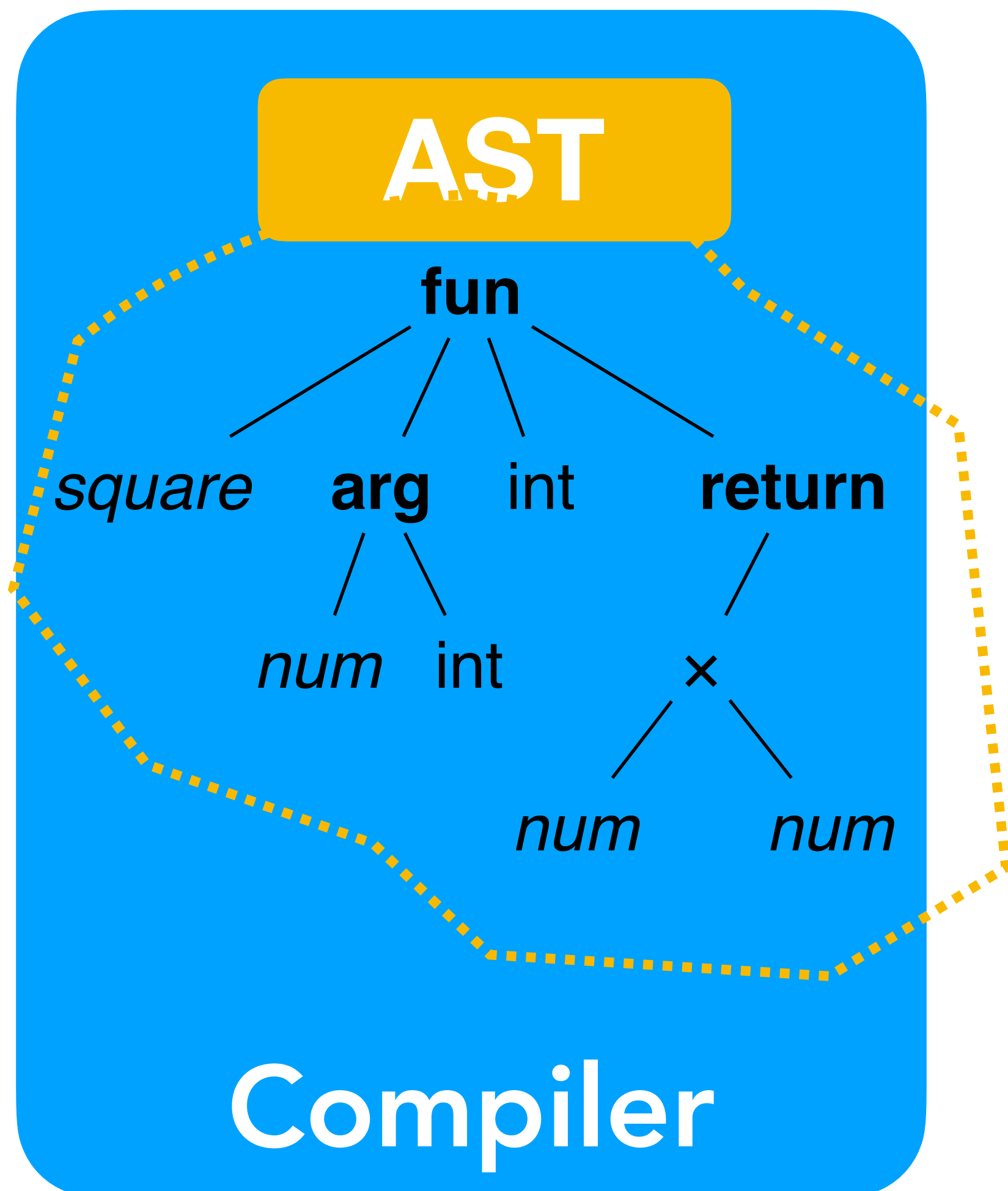


Abstract because we're not putting in every detail of the actual programming language syntax. (E.g., we've dropped all those pesky semicolons and parens.)

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 languages that have only one interpreter, and the interpreter is embedded in a 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.

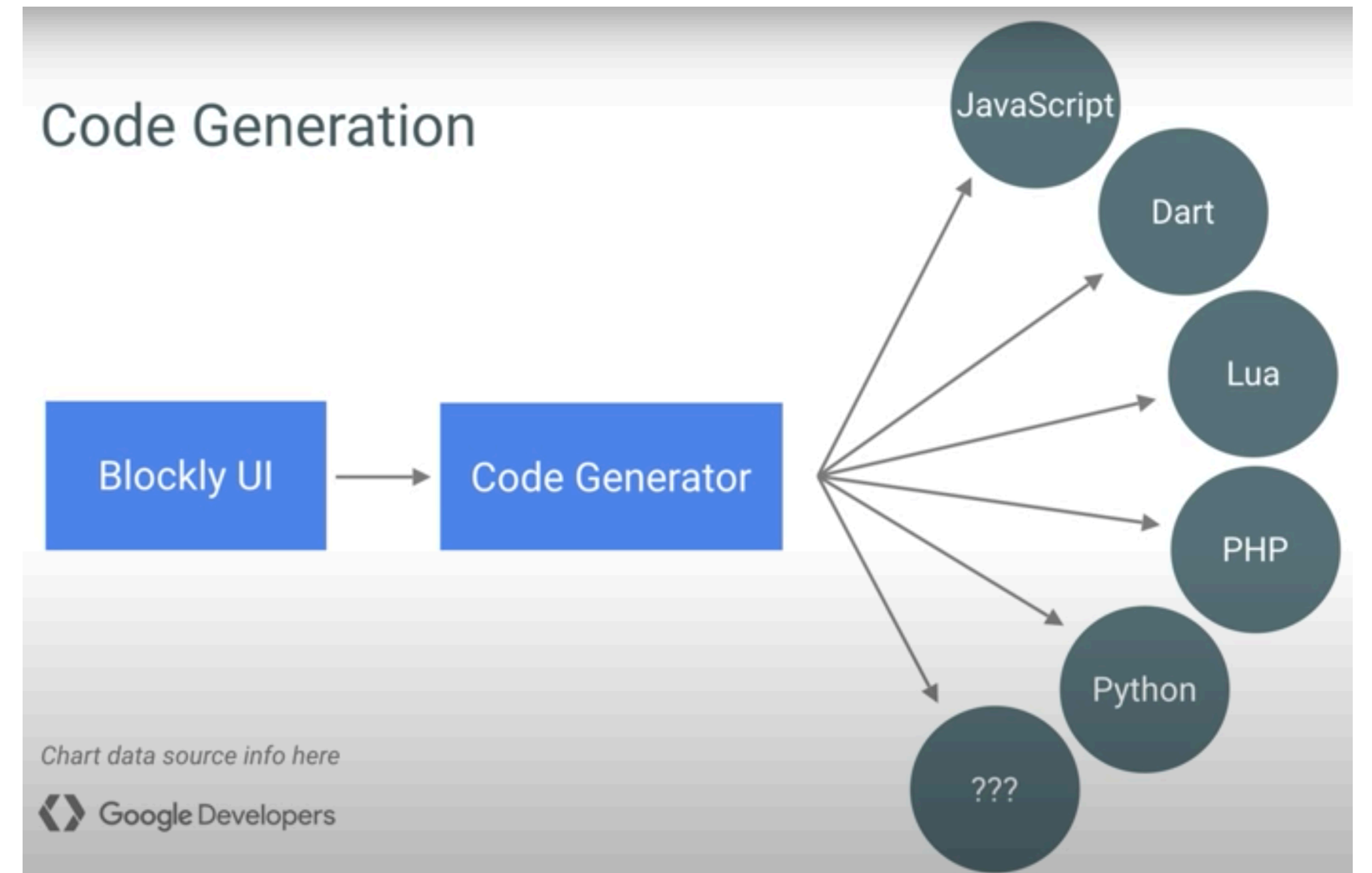
Programming Language vs. Programming Environment

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

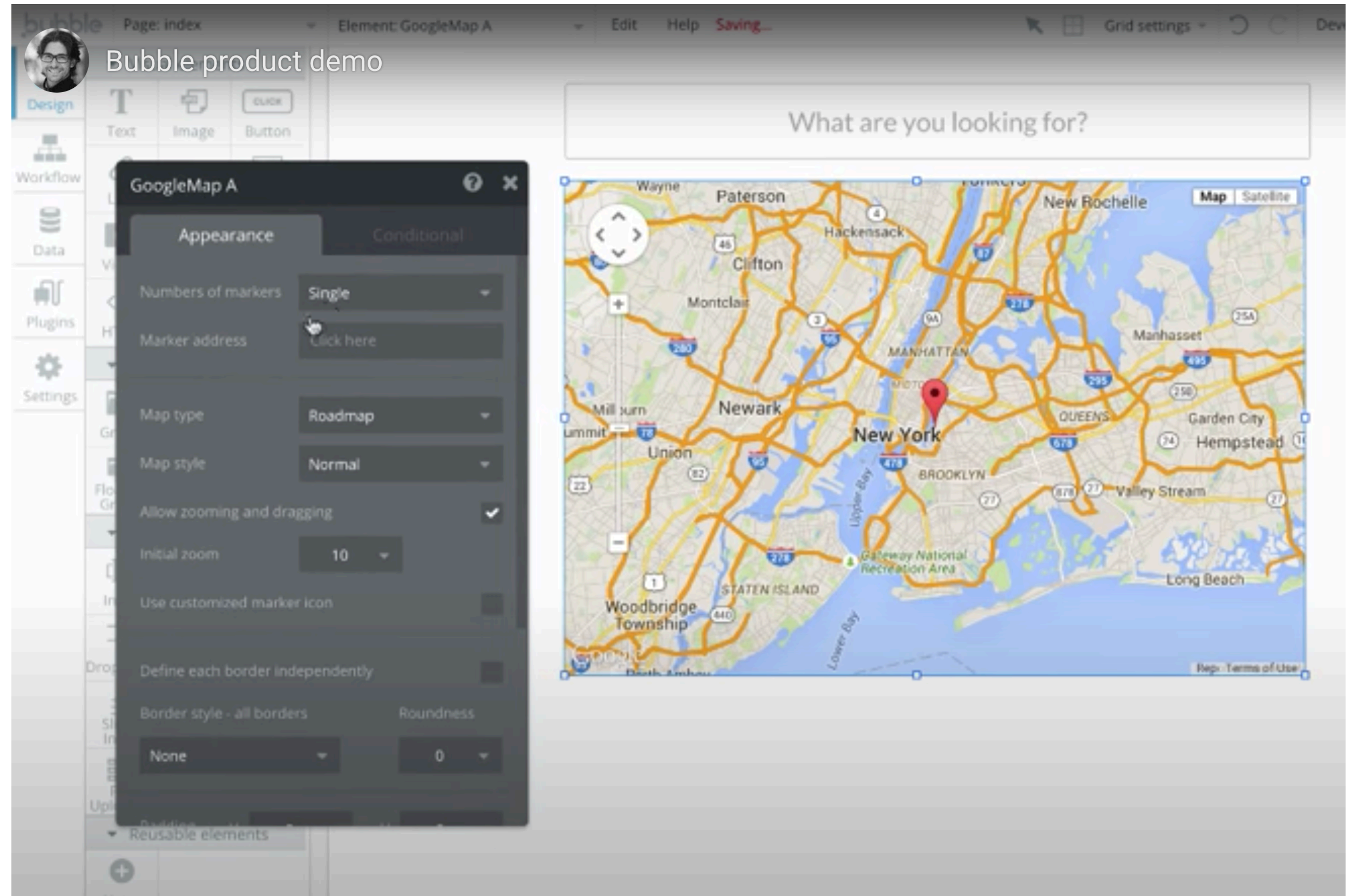
Projectional Editor: Any editor (can be textual or visual) in which we build up programs by interacting directly with ASTs

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

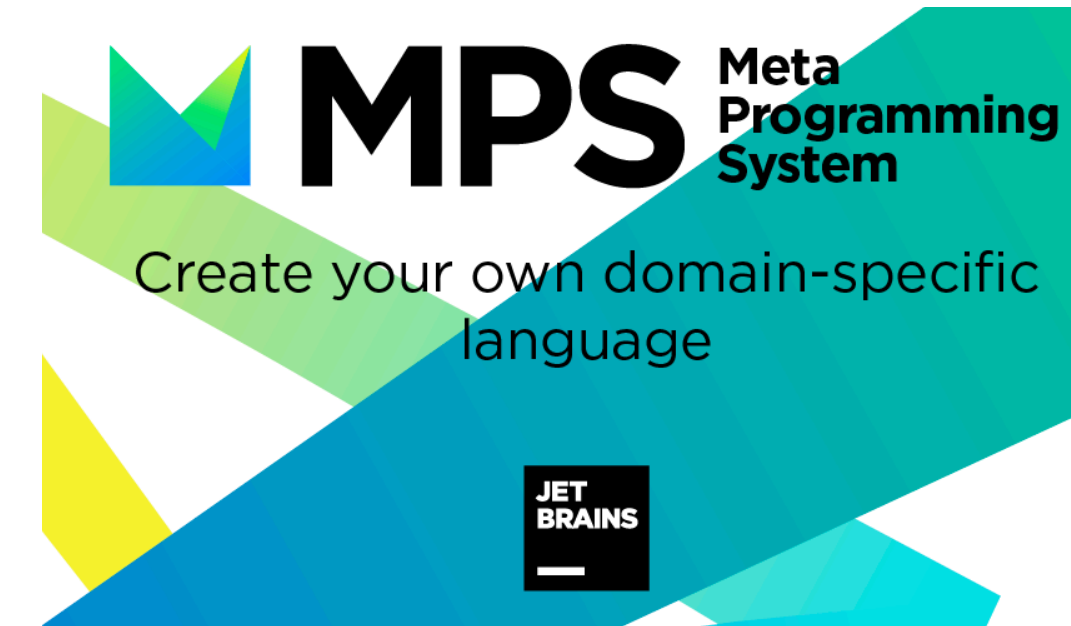


Dispose properly

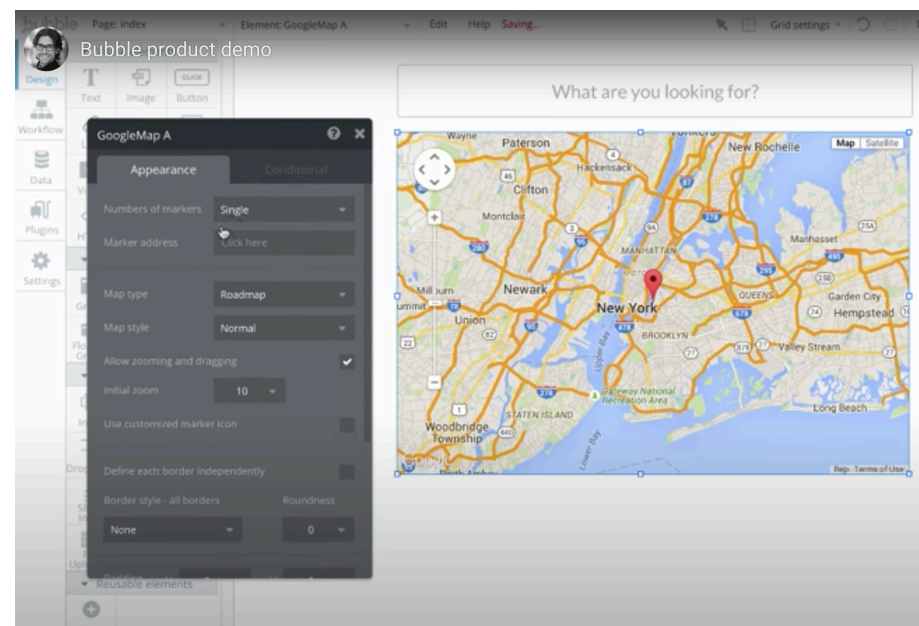
Non-Projectional

Projectional

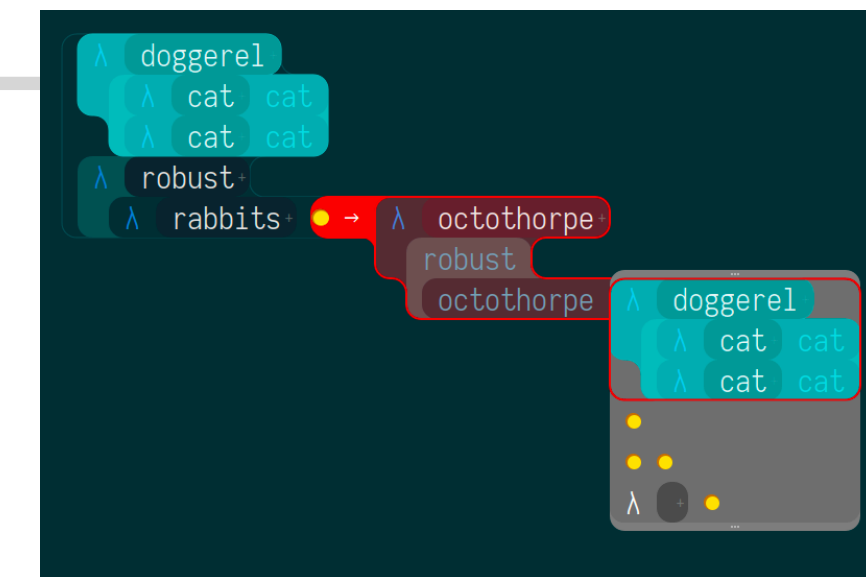
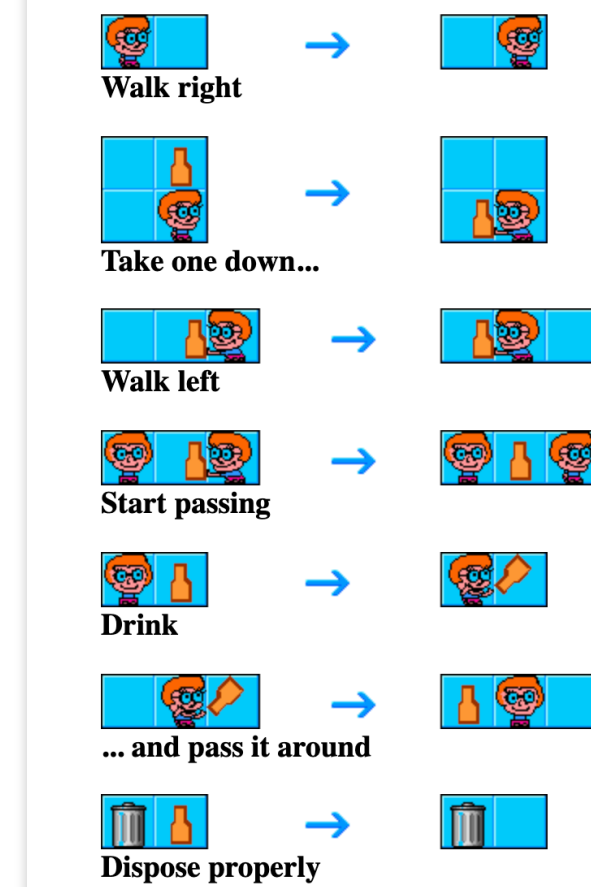
Textual



Visual



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



syntax errors vs. logical errors

Before we switch to activity time...

Reminder to be working on final
projects!!!

Snap! Activity

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

[https://snap.berkeley.edu/snap/help/
SnapManual.pdf](https://snap.berkeley.edu/snap/help/SnapManual.pdf)

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 languages intended to be used in 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

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 main workspace with a "Scripts" tab. The workspace contains a script for a "Sprite" object with the following blocks:

- set db to table
- add list bella 17 2 to db
- add list edward 104 1 to db

The left sidebar also shows a "Variables" section with a "table" variable and an "add to" block.

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 central workspace for a "Sprite" object. The "Scripts" tab is active, displaying a sequence of 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 bottom of the interface shows a "table" block with an "add" block and a "set column names of" block, indicating the start of a new table definition.

Snap! Activity - Stage 3

The screenshot shows the Snap! programming environment with a project titled "database_in_class_assignment". The left sidebar displays various categories: Motion, Looks, Sound, Pen, Control, Sensing, Operators, and Variables. The main workspace shows a script for a "Sprite" object, which is set to be "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 "table" block with an "add" block and a "set column names of" block. At the bottom, there is a "select column from" block.

A visual representation of the "names" list. It shows a list with two items: "bella" at index 1 and "edward" at index 2. The list has a length of 2.

Index	Value
1	bella
2	edward

A visual representation of the "ages" list. It shows a list with two items: "17" at index 1 and "104" at index 2. The list has a length of 2.

Index	Value
1	17
2	104

Reflection

- Was implementing database abstractions a difficult programming task? Hum for frustration level.
 - I'm so sorry. I know it's wildly frustrating
- If you're designing a new language, does putting it in a block-based editor automatically make it easy?
- Does getting out of syntax errors mean that you're not programming?
- It's probably been a while since you were a newcomer to programming, but this may have simulated some of that experience. If you're designing for newcomers in your final project, or for people who are uncomfortable doing the particular kind of programming for which you're designing, does this raise anything for you?

Snap! Activity - Stage 4

HW Assignment

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!)