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?

Assembly Language

```
1 // square a couple numbers
2 int square(int num) {
3     return num * num;
4 }
```
What’s happening inside my compiler?

What’s all this??

Assembly Language

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

Compiler

```
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
```
What’s happening inside my compiler?

Compiler

Parser

AST

Code Generator

Assembly Language

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

```
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
```
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
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
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.

<table>
<thead>
<tr>
<th>Action</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Walk right</td>
<td><img src="image" alt="Walk right" /></td>
</tr>
<tr>
<td>Take one down...</td>
<td><img src="image" alt="Take one down" /></td>
</tr>
<tr>
<td>Walk left</td>
<td><img src="image" alt="Walk left" /></td>
</tr>
<tr>
<td>Start passing</td>
<td><img src="image" alt="Start passing" /></td>
</tr>
<tr>
<td>Drink</td>
<td><img src="image" alt="Drink" /></td>
</tr>
<tr>
<td>... and pass it around</td>
<td><img src="image" alt="Pass it around" /></td>
</tr>
<tr>
<td>Dispose properly</td>
<td><img src="image" alt="Dispose properly" /></td>
</tr>
</tbody>
</table>
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
Snap! Activity

https://snap.berkeley.edu/snap/snap.html
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!
Goal for next reading

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