Meeting 05: Parsing

- HW1 Stats
  Hours spent: mean + median 12

- Release submitted tests?

```
Parsing
```
```
PO text -> compile, parser -> PO AST -> Flatten -> Flattened PO AST
```
```
HW2
```
```
x86 Assembly
```
```
Instrruct Selection
```
Parser

PS text → Lexer → Seq of tokens → "Actual" Parser

Sequence of characters

PLY

Lex | Yacc

(original UNIX)

Spec of "how to tokenize"

Spec of "how to parse"
spec for lex for building a lexer

Regular Expressions — matching strings

- \( L(c) \) = \{ "c" \}^3
- \( L(c) \) = \{ "c" \}^n
- \( r_1 r_2 \) (concatenation)

Ex: \( L(cc) = \{ "cc" \}^3 \)

- \( r^* \) (closure)
  - 0-or-more instances of \( r \)
\[ r_1 \mid r_2 \quad \text{("or", alternation)} \]

\[ \text{Ex}: \langle c \mid d d d \rangle = \{ "c", "d d" \} \]

\[ \varepsilon \quad \text{(empty)} \]

\[ \mathcal{L}(\varepsilon) = \{ "\" \} \]

\[ r^+ \quad \text{(one-or-more)} \]

\[ \text{def} \quad \mathcal{L}^+ = r \cdot r^* \]
re or finite automata

spec for lex (PLY)

re describes how to create tokens

regex `+`
Grammars (Context-Free)

Describing the syntax of our language

Creating structure

Describe 8 expressions

BNF

eexpr ::= eexpr \+ eexpr

|"\-"|\"\*"| eexpr

|INT | NAME | `(eexpr)`