CS61A

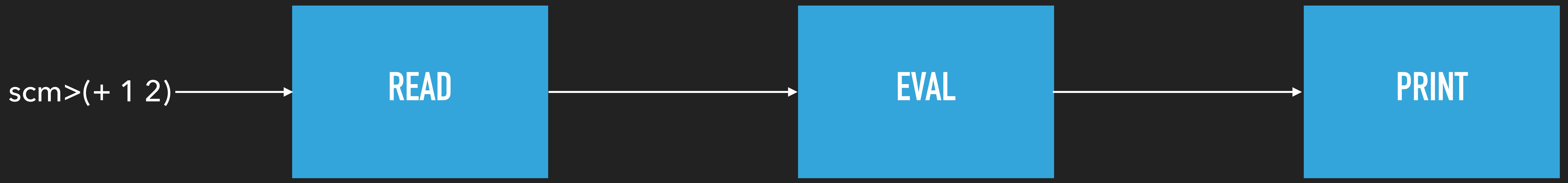
REGULAR EXPRESSIONS, BNF, SQL

LOGISTICS AND REMINDERS

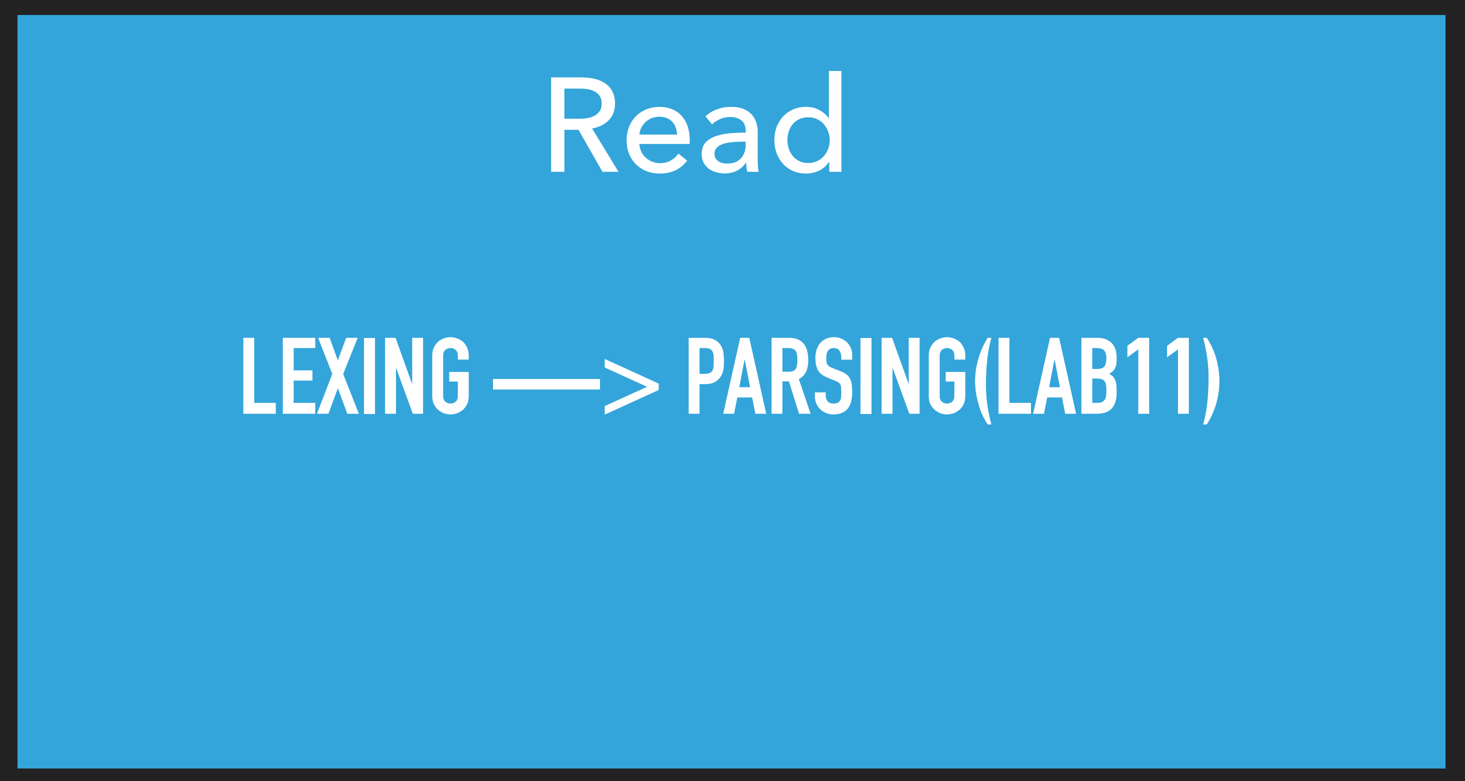
- HW9 due **Tomorrow**
- Scheme project due **next Tuesday**
  - Checkpoint 2 due **Friday**
- Lab12 due **Today**

RECAP OF LAB11: INTERPRETERS

- How does scheme interpreter work?(Read-Eval-Print-Loop)



- Read phase consists of **lexing**(creating tokens) and **parsing**(turning tokens into a data structure)



WHY DID WE DO THINGS LIKE THIS?

- Goal: Have valid scheme expressions parse properly, and incorrect things should error
- Attempt 1: RegEx
    - Matching sequential patterns of characters in code
    - Doesn’t allow us to parse ‘(), ‘(()), ‘((())), ‘(((()))), etc.[Think parenthesis problem]
        - In Python we can’t parse lists because RegEx can’t match [], [[]], [[[]]], [[[[]]]], etc.
- Attempt 2: BNF
    - More powerful than RegEx
    - Good enough generality to describe languages for us
- Lab11 was how we parse things, but it’s really one instance of handling BNF grammar just for Scheme!
- We want you to understand if language follows some constraints like BNF grammar how and why we parse the way we do.

REGEX(ATTEMPT 1 AT UNDERSTANDING LANGUAGES)

- Goal: Let's try and find a concise way to identify if we have an input that is included in some set of possible inputs typed in(call this inclusion a match)
- What sets?
  - "A", "AA", "AAA", "AAAA", etc
    - infinitely many items, but are there patterns?
  - 732-234-2134, 201-923-0312, 510-120-1293, 023-421-421, etc
- Ideally if we can identify if we have some item in a predefined set we can write an interpreter!
  - Ex. (+ 1 2), (+ 1 3), (* 1 4), …
    - If we can identify call expressions and get arguments then we write code to process them in the same way

REGEX — SIMPLEST SETS

- Single characters!
- How do we find out if we are in this set(ex. “A”)?
  - Regex Pattern: r”A”
  - Just see if our pattern has a single A
- Unfortunately: some characters in Regex are special(ex. ( ) . + *)
  - To find out if we are in the set (ex. “(“)
    - Regex Pattern r”\(“
- Now we can handle all single character sets

REGEX — CHARACTER SETS WITH MORE THAN ELEMENT

- How do we match a set consisting of finitely many single character elements?
  - "0", "1", "2", "3", "4", "5", "6"
- Regex Character Classes
  - An input can match any single character in a given class of characters
  - Regex Pattern: r"[0123456]"
- Just put whatever you want in a square brackets and that will represent the set of single characters

REGEX- MULTIPLE CHARACTER SETS

- “aa”, “ab”, “ac”, “bb”, “bc”, “ba”, “ca”, “cb”, “cc”
- Regex Pattern: r“[abc][abc]”
- Idea: list consecutive characters or character classes that you want to match
- Why is “aa”, “bb”, “cc”, “dd”, “ee”, etc. harder?
- What are current restrictions based on things we’ve defined?

REGEX-INTEREST IN THE INFINITE

- What about infinite sets: “a”, “aa”, “aaa”, “aaa”, “aaaa”, “aaaaa”, etc.
- It would be nice to have a way to specify quantity!
- Regex has quantifiers to do this: *, +, ?, {1,}
    - Regex Pattern: r”a+” matches 1 on more a
    - Quantifiers only affect single character or character class
    - * is 0 or more characters –>allows us to match infinite
    - + is 1 or more characters –> allows us to match infinite
    - ? Is 0 or 1 of that character
    - {i,j} is from i to j amount of that character

CONVENIENCES AND SHORTHANDS

- We have some predefined character classes:
- \d corresponds to digits 0-9
- \w corresponds to alphabet(upper or lowercase), digits, _
- \s matches spaces and whitespace
- . matches any character that is not newline
- Is [0123456789] the same as \d?
- [0-9] is shorthand for [0123456789], and [a-z] for [abcdedfghijklmnopqrstuvwxyz]
- How do we do include dashes? Escape the dash(\-) or put it first or last in character class

LAST THING: ANCHORS

- There are three: \b, ^, $
- ^ means should start at the beginning of string
- $ should be at end of string
- \b is more complicated corresponds to word boundaries
  - Keeps track of changes from word characters to not word characters, not word characters to word characters, start/end of string
  - NOT A character class
- USE regex101.com it's a lifesaver!

REGEX FAILS PARTS OF PROGRAMMING LANGUAGES

- We mentioned earlier no regex can match the set
  - [], [[]], [[[]]], [[[[]]]], etc.
  - ‘(), ‘(()), ‘((())), ‘(((()))), etc.
- Regex isn’t suitable to describe programming languages

BNF(BACKUS-NAUR FORM)

- Solution to our problem, gives us enough generality to parse languages of interest
- How do we define what should be allowed to match in this new things?
    - Define a set of rules
        - If input follows rules then we can parse it!

BNF(SYNTAX FOR DEFINING RULES)

symbol0: symbol1 symbol2

Symbols represent sets of strings

- Symbols can be strings or regex(terminal symbols)(everything we could do before)
- Symbols can be defined in terms of terminal symbols or non-terminal symbols
    - Secret Sauce to making BNF more powerful than Regex

EXAMPLE BNF

parens: “[]” | “[“ parens “]”

This corresponds to the: [], [[]], [[[]]], [[[[]]]], etc

So we have defined a set of strings that will be matched properly and additionally parsed into a useful structure!

Let’s look at some parse trees! Can we expand this to the a larger set of nested brackets. Let’s go to code.cs61a.org

AMBIGUOUS BNF GRAMMARS

- We can now find matches!
- However multiple ways to parse can exist and this is bad?
- Can we construct an example of this?

REPRESENTATION

- BNF has multiple representations
  - The set of rules defining a grammar
  - The set of all strings parsable by the grammar
  - A railroad diagram

SQL

- Manipulating tables
- SELECT columns FROM table WHERE condition;
- Can have multiple conditions joined with AND
- Multiple columns via comma separation
- Nothing else tricky today!