Welcome to 61A Lab!

We will begin at **5:10**!

Slides: **cs61a.bencuan.me**

Announcements

- HW6 due Thursday

The Plan

- Interpreters!
- Lab 11 Walkthrough
  - This lab is kind of hard ngl

Interpreters

The Calculator Example

- **Goal**: let's write an interpreter that understands simple math expressions!
  - (+ 2 2)
  - (- 5)
  - (* (+ 1 2) (+ 2 3))
- Understands +, -, *, /, and nested expressions

Pairs

- Pairs are literally linked lists!! Main differences:
  - **Pair** vs Link
  - **nil** vs Link.empty
  - **Pair(1, nil)** vs **Link(1)**
- Used to represent Scheme code in Python

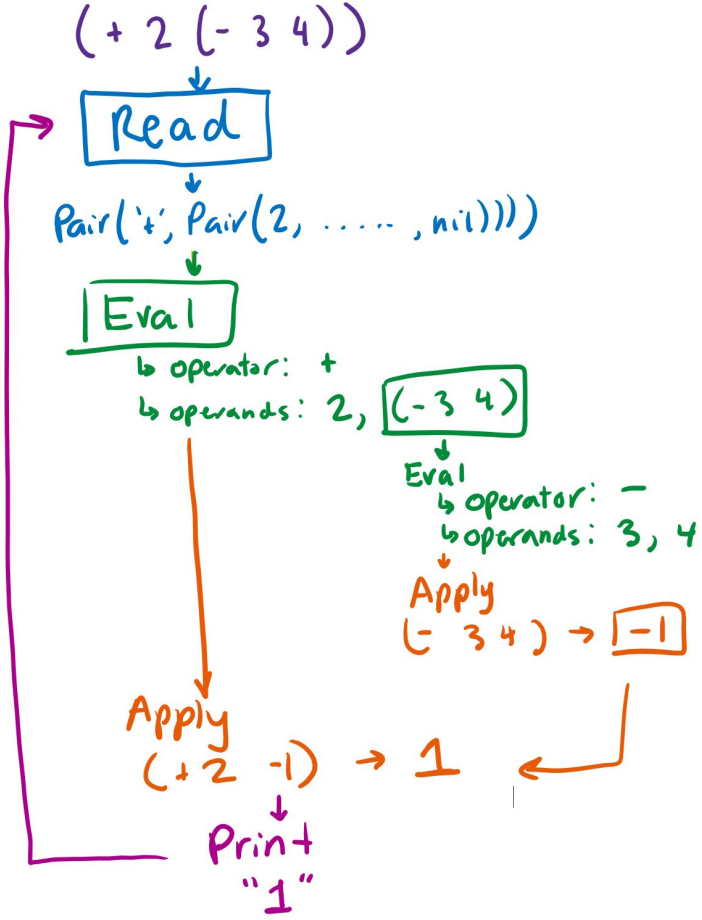
Operators and Operands

- An **operator** is the function you are trying to apply in Scheme
  - +, list, append...
  - The **very first** element in a Pair list
- An **operand** is a parameter that is passed into the function
  - In (+ 3 5), + is the operator and 3, 5 are both operands

Eval and Apply

- **Eval rules:**
  - Basic elements (numbers, booleans...) - done (base case)
  - Built-in functions (+, -, ...) - lookup in OPERATORS
  - Function:
    - Call eval on operator
    - Recursively call eval on all operands
    - Call apply on operator and operands

Example


(+ 2 (- 3 4))
Read
Pair('+', Pair(2, ... , nil)))
Eval
↳ operator: +
↳ operands: 2, (- 3 4)
Eval
↳ operator: -
↳ operands: 3, 4
Apply
(- 3 4) → -1
Apply
(+ 2 -1) → 1
Print
"1"

Lab Walkthrough

Today's Task, summarized

- Create the first part of the Scheme project!
- Part 1: Lexical Analysis (buffer.py)
  - Break down a long string into tokens
- Part 2: Syntactic Analysis (scheme_reader.py)
  - Evaluate tokens into final result

The Buffer Class

- A way to access tokens one at a time
- current: the next token
- pop_first(): removes next token from list, then returns it
- end_of_line(): returns true if current token is EOL_TOKEN
- If nothing left in buffer, pop_first() and current are None

Editing the Buffer Class (Problem 1)

- create_generator():
  - yields one token at a time from source **iterator**
- __init__():
  - Create a new generator (create_generator)
  - Initialize current token
- pop_first():
  - Use generator made in __init__
  - Reassign self.current and return the old one

Scheme Read (Problem 2)

- Goal: convert **Buffer** of tokens into a **Python representation**
  - **Mutual recursion:** scheme_read calls read_tail, which calls scheme_read, which calls read_tail...
- Base case: returns a single token (like 5 or nil)
- Recursive case: pass src into read_tail

Read Tail (Problem 2)

- Goal: Read the **rest** of a valid Scheme expression
  - Example inputs: `), + 2 3),`
  - These inputs are not valid: `(+ 1 2), nil`
- Base case: return nil if at a closing parentheses)
- Recursive case:
  - scheme_read first
  - read_tail rest
  - return a pair of first and rest

Read Quotes (Problem 3)

- Goal: Add handling of inputs like `(quote (+ 1 2))` or `‘(+ 1 2)`
- Modify scheme_read to add a new quote case
  - If quote detected, create a Pair containing the literal quote char and a **nested Pair** containing the rest

Lab Hints

Work Time!

go.cs61a.org/ben-queue