LAB 11

~~Interpreter~~

Project 4: Scheme Interpreter

Mingxiao Wei
mingxiaowei@berkeley.edu

Nov 07, 2022

LOGISTICS 🏡

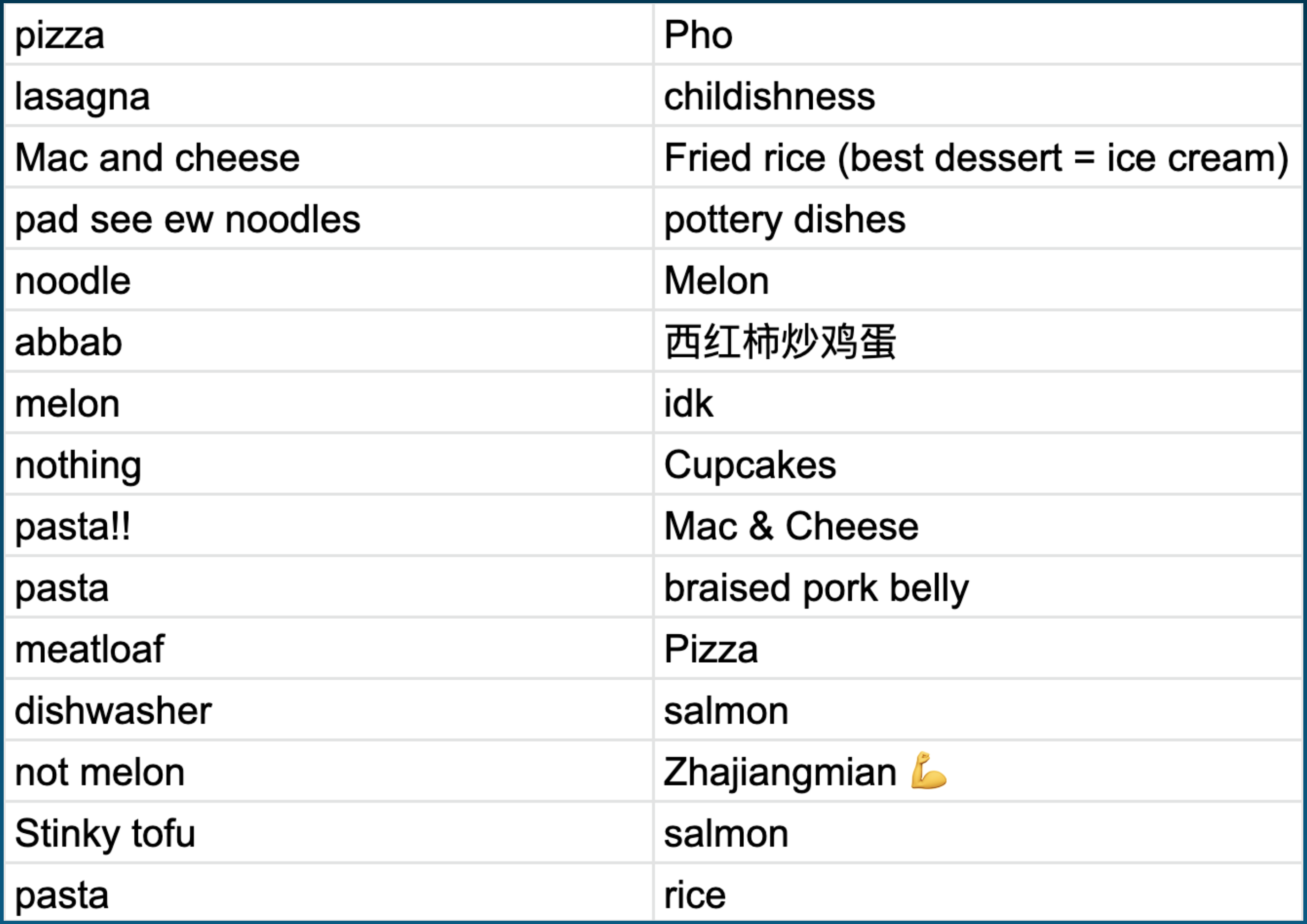
- Lab 11 is now optional - everyone receives full credit for the assignment 🤑
- Project 4 released!
    - Checkpoint 1 due tomorrow 11/08
    - Checkpoint 2 due Sun 11/13
    - The whole project due Tue 11/22
    - Everyone gets 2 EC points for free 👐
    - One EC problem, worth 1 EC point 👍
    - Submit the whole project by Mon 11/21 for 1 EC point 💪
- Homework 08 due next Thu 11/17

ABOUT THE STRIKE

- Why Strike
- Unfair Labor Practice Charged Filed Against UC
- EECS Staffing Bargaining Proposal
- How Can Y'all Support
- 61A-wise Logistics

FROM LAST TIME... 👀

What's the best dish?



pizza	Pho
lasagna	childishness
Mac and cheese	Fried rice (best dessert = ice cream)
pad see ew noodles	pottery dishes
noodle	Melon
abbab	西红柿炒鸡蛋
melon	idk
nothing	Cupcakes
pasta!!	Mac & Cheese
pasta	braised pork belly
meatloaf	Pizza
dishwasher	salmon
not melon	Zhajiangmian 💪
Stinky tofu	salmon
pasta	rice

INTERPRETER 📥

INTERPRETERS OVERVIEW

- Interpreter - A program that interprets and processes other programs
  - In project 4, we'll build an interpreter for Scheme using Python
- Read-Eval-Print Loop (aka REPL)
  - Read - Parse Scheme programs into Python representations
  - Eval - Evaluate the Scheme program (now in their corresponding representation in Python) using Python
  - Print - Print out the evaluation result

INTERNAL REPRESENTATIONS

Scheme	Python
Numbers	Python's built-in `int` and `float` values
Symbols	Python's built-in `string` values
Booleans (`#t`, `#f`)	Python's built-in `True`, `False` values
Combinations (lists, call expressions, special forms)	Instances of the `Pair` class, defined in `pair.py`
`nil`	The `nil` object, defined in `pair.py`

THE PAIR CLASS

- Full definition in `pair.py`
- Similar to a linked list - each instance has `first` and `rest` attributes
- Empty pair - `nil`, an object, not a class attribute
- We can call `len()` on a `Pair` object to get its length
- The `map(fn)` method maps the **one-argument** function `fn` to each element, and returns a new `Pair` object (non-mutative)

```
>>> p = Pair('+', Pair(1, Pair(2, nil)))
>>> p.first
'+'
>>> p.rest
Pair(1, Pair(2, nil))
>>> p.rest.first
1
>>> p.rest.rest.rest is nil
True
```

PARSING SCHEME COMBINATIONS INTO PAIRS

- A Scheme combination `(a b c)` is parsed into a `Pair` object `Pair(a, Pair(b, Pair(c, nil)))`
  - `a`, `b`, and `c` can be *anything* - symbol, number, another combination, etc
- This happens in the read stage - <u>no evaluation yet</u>, everything is either a number, a boolean, or a string (in Python)
- Done by `read_line` (implemented already)

```
>>> read_line( '(+ 1 2)' )
Pair('+', Pair(1, Pair(2, nil)))
>>> read_line( '(define x 3)' )
Pair('define', Pair('x', Pair(3, nil)))
>>> read_line( '(length (list 6))' )
Pair('length', Pair(Pair('list', Pair(6, nil)), nil))
>>> read_line( '(cons 4 (cons 5 nil))' )
Pair('cons', Pair(4, Pair(Pair('cons',
Pair(5, Pair('nil', nil))), nil)))
```

PRO TIPS FOR THE PROJECT

- Get started early!
- READ THE SPEC - we are really trying to tell you what to do
- Consult the Getting Started Videos if you're not sure where to start!

ATTENDANCE! 🤠

go.cs61a.org/mingxiao-att

- The attendance form and slides are both linked on our section website!
- If you finish early, let me or any of the AI's know and we'll check you off
- Once again, please do remember to fill out the form by midnight today!!