CS 131 Discussion

Week 3: Project Pals

matt wang (he/him)
✉️ matt@matthewwang.me
🔗 mattxw.com/131slides
📣 mattxw.com/131feedback

Discussion Agenda

1. feedback from last week
2. warmup
3. selected hw answers
4. person of the week

   ~ break ~

5. language of the week
6. project deep dive
7. open Q&A

snacc of the week
(actually)

Feedback / Iteration

Thank you for giving feedback!

Some things I'm working on this week:

- **pacing**
  - **less hw review**
- first online OH!
- project walkthrough (incl setup, tooling)

Back of mind:

- PDF export for slides / notes – working with Carey

warmup

(problem solving strategies)

how would we merge two (sorted) lists in Haskell?

```haskell
merge [1,5,10] [2,4,8] = [1,2,4,5,8,10]
```

to start: trivial base cases. what are they?

```
merge [1,5,10] [2,4,8] = [1,2,4,5,8,10]
```

great! what's the single pattern match step?

```
merge [] a = a
merge a [] = a
```

great! what's the single pattern match step?

```
merge [] a = a
merge a [] = a
merge a@(ah:as) b@(bh:bs)
```

great! what's the single pattern match step?

```
merge [] a = a
merge a [] = a
merge a@(ah:as) b@(bh:bs)
  | ah < bh   = ah : merge as b
```

great! what's the single pattern match step?

```
merge [] a = a
merge a [] = a
merge a@(ah:as) b@(bh:bs)
  | ah < bh   = ah : merge as b
  | otherwise = bh : merge a  bs
```

what's the time complexity?

```haskell
merge [] a = a
merge a [] = a
merge a@(ah:as) b@(bh:bs)
  | ah < bh   = ah : merge as b
  | otherwise = bh : merge a  bs
```

last week’s homework!

(abridged)

People seemed to struggle with...

- Question 1b - “list of lists”
- Question 3b - `a -> b -> c, (a -> b) -> c, a -> (b -> c)`
- **Question 4**
- Question 9 - list fibonacci
  - Suggestion: inefficiency is fine, if we don’t ask for efficiency!
- Question 10 - Super Giuseppe
  - In particular, people’s solutions were *way too verbose*

Suggestion: review these ones in particular when it’s midterm time!

Question 4

```haskell
f a b =
  let c = \a -> a    -- (1)
      d = \c -> b    -- (2)
  in \e f -> c d e   -- (3)
```

This question was a lil' bit of a struggle … let's walk through it together!

Question 4

```haskell
f a b =
  let c = \a -> a    -- (1)
      d = \c -> b    -- (2)
  in \e f -> c d e   -- (3)
```

Which `a` is this?

Question 4

```haskell
f a b =
  let c = \a -> a    -- (1)
      d = \c -> b    -- (2)
  in \e f -> c d e   -- (3)
```

Shadowing!

Question 4

```haskell
f a b =
  let c = \a -> a    -- (1)
      d = \c -> b    -- (2)
  in \e f -> c d e   -- (3)
```

What is the type of `c`?

Question 4

```haskell
f a b =
  let c = \a -> a    -- (1)
      d = \c -> b    -- (2)
  in \e f -> c d e   -- (3)
```

What does `c` do?

Question 4

```haskell
f a b =
  let c = \a -> a    -- (1)
      d = \c -> b    -- (2)
  in \e f -> c d e   -- (3)
```

What we know:

- `c` just returns its input

What does `d` do?

Question 4

```haskell
f a b =
  let c = \a -> a    -- (1)
      d = \c -> b    -- (2)
  in \e f -> c d e   -- (3)
```

What we know:

- `c` just returns its input
- `d` **always** returns `b`

What does `c d e` do?

Question 4

```haskell
f a b =
  let c = \a -> a     -- (1)
      d = \c -> b     -- (2)
  in \e f -> c d e    -- (3)
```

What we know:

- `c` just returns its input
- `d` **always** returns `b`
- `c d e` **always** returns `b`

What is the type of `f`?

Question 4

```
f a b =
  let c = \a -> a     -- (1)
      d = \c -> b     -- (2)
  in \e f -> c d e    -- (3)
```

What we know:

- `c` just returns its input
- `d` **always** returns `b`
- `c d e` **always** returns `b`
- `f` returns a 2-arg lambda

What is `f 4 5`?

Question 4

```haskell
f a b =
  let c = \a -> a     -- (1)
      d = \c -> b     -- (2)
  in \e f -> c d e    -- (3)
```

What we know:

- `c` just returns its input
- `d` **always** returns `b`
- `c d e` **always** returns `b`
- `f` returns a 2-arg lambda

What is `f 4 5 6 7`?

Question 9

Write a Haskell function named fibonacci that takes in an `Int` argument `n`. It should return the first `n` numbers of the Fibonacci sequence.

Examples:
`fibonacci 10` should return `[1,1,2,3,5,8,13,21,34,55]`.
`fibonacci -1` should return `[]`.

Hint: You may find it easier to build the list in reverse in a right-to-left manner, then use the reverse function.

Solution 1: “Naive”

```
-- second_last is O(n), ++ is O(n); total is O(n^2)
fibonacci 1 = [1]
fibonacci 2 = [1, 1]
fibonacci n =
    let second_last xs
          | length xs == 2 = head xs
          | otherwise = second_last (tail xs)
        prev_fib = fibonacci (n-1)
    in prev_fib ++ [last prev_fib + second_last prev_fib]
```

Solution 2: Reverse-build

```
fibonacci :: Int -> [Integer]
fibonacci n =
  let fib_rev 1 = [1]
      fib_rev 2 = [1, 1]
      fib_rev n =
        let prev_fib_rev = fib_rev (n-1)
            first = head prev_fib_rev
            second = head (tail prev_fib_rev)
        in (first + second) : prev_fib_rev
  in reverse (fib_rev n)
```

Solution 3: Infinite (we do not expect this!)

```
-- fancy solution using list comprehension that
-- generates an infinite list.
-- this works because of Haskell’s lazy evaluation.
fibonacci n =
  let fib = 1 : 1 : [a+b | (a,b) <- zip fib (tail fib)]
  in take n fib
```

overview: this week’s HW!

- python syntax, basics
- FP in Python
- OOP in Python

Not all the questions are mandatory!

person of the week

🚧🚧🚧🚧🚧

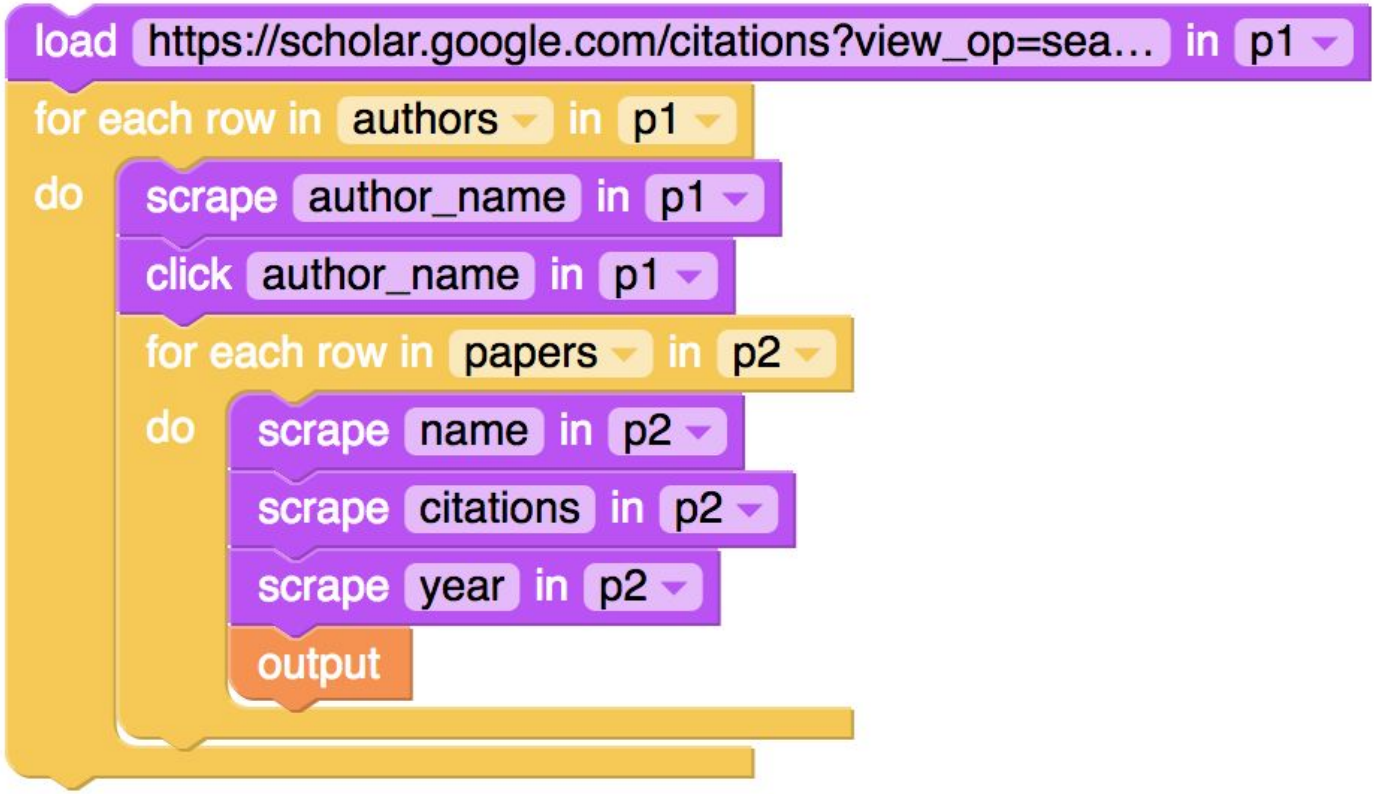
Sarah Chasins (~1990-)

- Prof in EECS at Berkeley
- Research:
  - Programming Languages
  - PL + Human-Computer Interaction · Program Synthesis · **Programming Tools for Social Sciences**

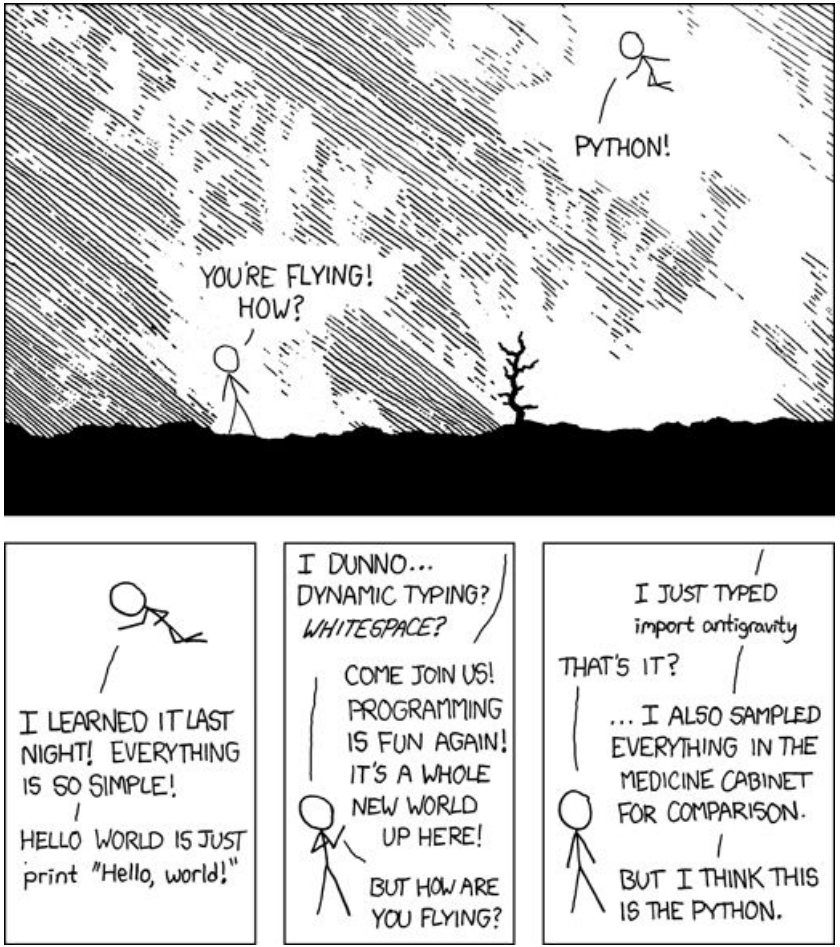
Why Sarah Chasins?

- We can get *more general purpose than Python!*
- PL isn't only
  - for CS people!
  - used by CS people!
  - stuff that's edited in emacs
- Matt thinks her research is cool :)

Sarah Chasins!
Her website, unknown.



Helena demo. [Helena homepage](), 2017.



~ break ~

discussion will resume at 10:55

language of the week

Python variants

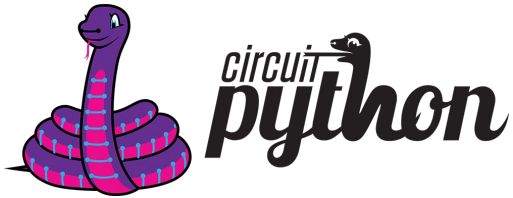
(okay, it's not one language)

Jython

Python on JVM

PyPy

JIT Python



CircuitPython

For microcontrollers!

Why talk about variants?

Common misconception: only one implementation of languages!

Important: **separation of interface and implementation.**

Case study, PyPy:

- "On average, PyPy is 4.8 times faster than CPython"
- Stackless Python: *erase call stack on function calls!!!*
    - massive concurrency benefit
- memory blowup with lack of reference counting
- lack of compat with some C FFI / extensions

And, *the **failure** of certain separations*!

- Libraries are over-reliant on features *not in the Python spec*, but part of CPython!
- Big issue with Python 2.x -> Python 3.x!

project overview / walkthrough

Useful Project Links:

- Spec
- Gradescope
- Template + Autograder (optional)
- Tips (from Ashwin!)

What's Different?

Different from other UCLA CS projects...

- **instant feedback**
  - 20% / 10 test cases are instantly graded and visible
- **public test cases (subset)**
  - same test cases are on GitHub (autograder repo)
- local autograder
  - imo, just fun :)

Bottom line: **I feel very passionate about transparent grading.**

Please give us feedback!

Prerequisites

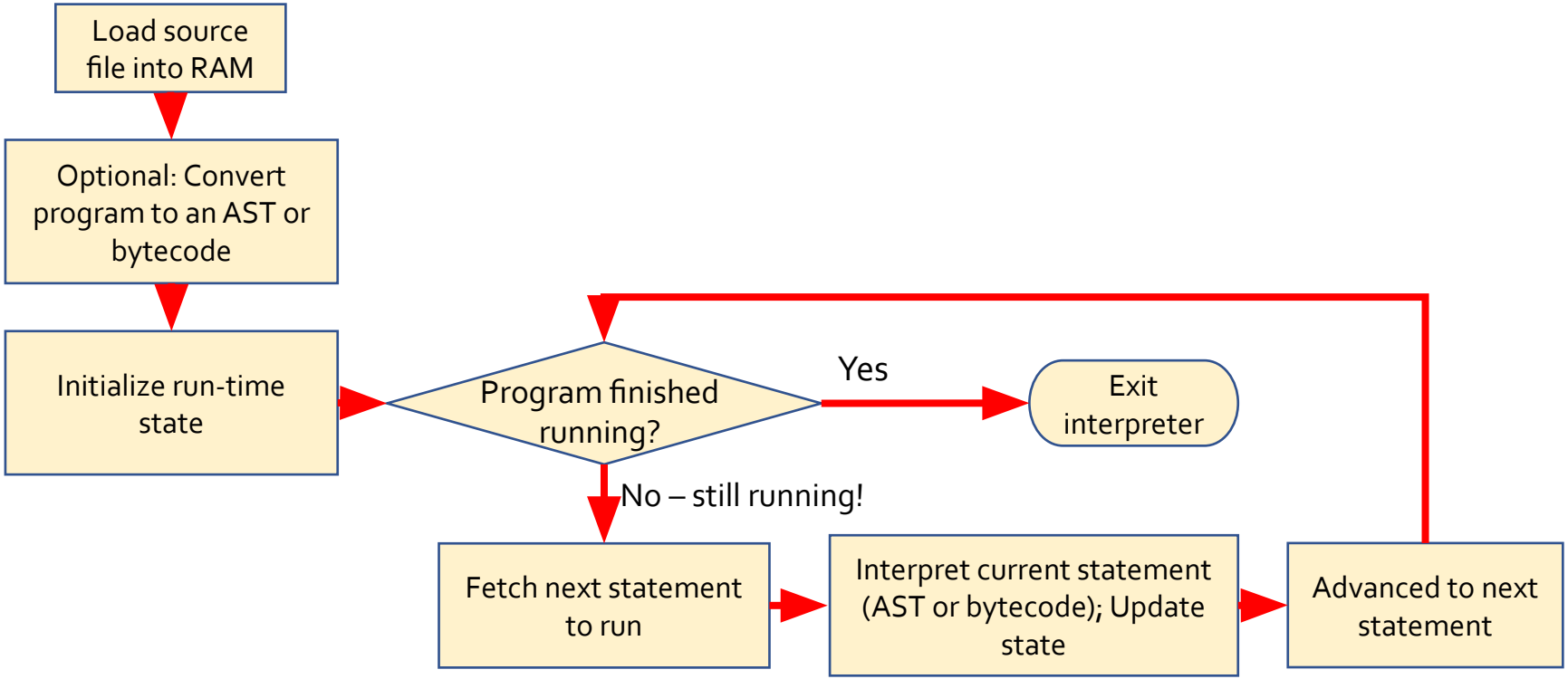
On your computer:

- **Python 3.x** (we are going to grade on 3.10, but ~3 is good)
- recommended: an editor with a Python extension
  - ex: Pylance on VSCode

Prerequisites

In your ~ brain ~

- Python syntax / basics
  - Syntax: control flow, types
  - OOP in Python *(it's different!)*
  - Being "Pythonic": the standard library, data structures
- Mental model of an interpreter - next slide :)
- The spec! Test cases!



From Carey's slides!

Interpreter Intuition

Big picture problems:

- What needs to be done *before* you interpret line-by-line?
  - Lexing / tokenization, finding “start” / main function
  - When do you remove comments?
- What state do you need?
  - “pause” the program, and pick up later – what do you need to continue?
  - What do you *not* need?

```
000 # Our first Brewin program!
001 func main        # our main function
002    funccall input "Enter a number: "
003    assign str_val result
004    funccall strtoint str_val
005    assign n result
006    funccall factorial
007    funccall print result
008 endfunc
009
010 func factorial   # compute a factorial
011  assign f 1
012  while > n 1
013    assign f * f n
014    assign n - n 1
015  endwhile
016  return f
017 endfunc
```

versus other languages

Similar to Python-adjacent languages

- dynamically-typed!
- no type annotations!
- indentation-based
- identifiers: case-sensitive, alphanumeric-ish
- no type coercion

versus other languages

Different from most languages:

- only global variables!
- infix notation for operators
- no semicolons, parens, etc. – no precedence operators!
- no explicit `None` / `nil` / `nullptr`
- return is assigned to result
    - note: this *doesn't happen if there's no/blank return*

Suggestion: top-level interpret

Many interpreters have a function that looks like this:

```
interpret_tokenized_line(tokens):
    if tokens[0] is function_call:
        handle_function_call(line)
    else if tokens[0] is while:
        handle_while(line)
    else if tokens[0] is endwhile:
        handle_endwhile(line)
    ...
```

Suggestion: top-level interpret

Many interpreters have a function that looks like this:

```
interpret_tokenized_line(tokens):
    if tokens[0] is function_call:
        handle_function_call(line)
    else if tokens[0] is while:
        handle_while(line)
    else if tokens[0] is endwhile:
        handle_endwhile(line)
    ...
```

These handlers should be **self-contained**, and many of them should be pretty short!

“Open” Questions: Types

Do we need type information?

- 🤔

“Open” Questions: Types

Do we need type information?

- Yes, since there are type errors?

Do we need to explicitly store type information?

- 🤔

“Open” Questions: Functions

Do we need to worry about variable scopes?

- 🤔

“Open” Questions: Functions

Do we need to worry about variable scopes?

- No! All variables are global.

So, when we call a function, what do we need to keep track of?

- 🤔

“Open” Questions: Operators

What does this do?

```
== str1 “foobar”
```

What about this?

```
== var1 - var2 5
```

Or, this?

```
& > var1 5 <= var2 3
```

Closing Thoughts

My own ~ tips ~

- **Start early!!!!!!!!!!!**
- Don’t implement the entire solution at once; test small chunks!
  - Ex: can you get just variable assignment and printing to work?
  - You can start with an *incorrect subset of the language*!
- Ask for help :)

~ post-discussion survey ~

always appreciate the feedback!

- how is pacing?
- did we like less hw review?
- how helpful was project stuff?

see you next week <3

https://forms.gle/33gPkKDfajrrQrZ88