Today

- More compiles vs interpreters
- Tokenizer assignment
- CoK #2 Friday
- Grammars and BNF

How do you build an intepreter?

① Tokenizer / Scanner / Lexer

job of tokenize is to read program text and split into "atomic" meaningful units in the language ↷ lexemes

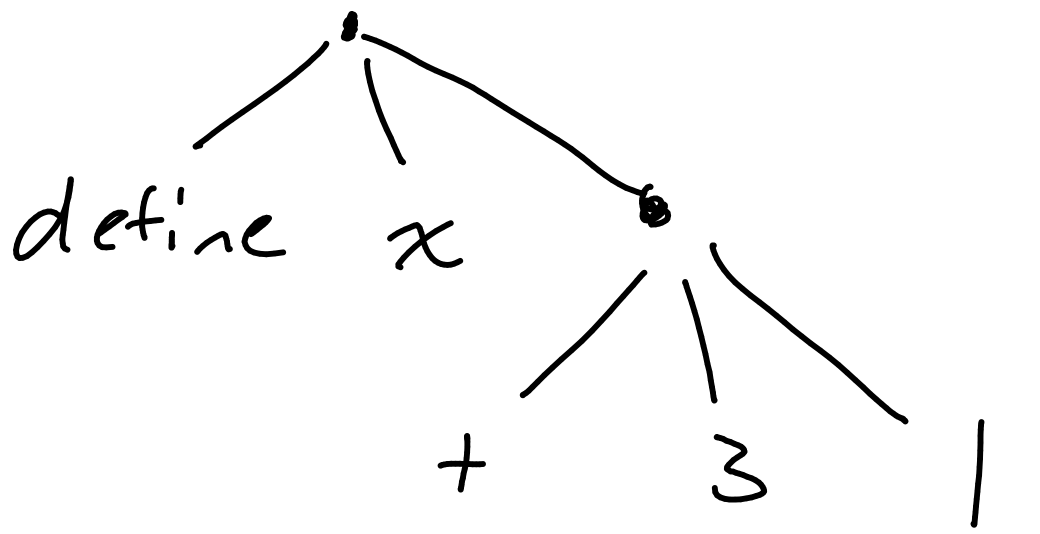
$\underbrace{(}\,\underbrace{\text{define}}\;\underbrace{x}\;\underbrace{13}\,\underbrace{)}$

lexemes	tokens - lexemes labeled by type	
(	(	left paren
define	define	string
x	x	string
13	13	int
)	)	right paren

② Parser

Identify <u>structure</u> of program - build a tree out of program

(define x (+ 3 1))



③ (...?) Interpreting

- executing code in the tree that you built

Timing

- Tokenize is a big one.

M	4/28	talloc	M	5/5	midterm break
W	4/30	work on tokenizer	W	5/7	tokenizer due
F	5/2	CoK	F	5/9	

Compilers / interpreters

. It is often said that interpreters process a program one line at a time, translating each line and then running it. While an interpreter could be designed to work that way in some very limited circumstances, this is in general not true. Why is it less than accurate to say that an interpreter is a system that translates and runs code line-by-line?

This is an oversimplified myth.

Why?

- What is a line of Scheme, anyway?

```
(car (cdr (map (lambda () ...))))
```

Go back to "greet"

```
hello     3
fun       4
you       2
suppress
```

New feature.
Any time "suppress" appears anywhere in the program, nothing gets printed.

- a line-by-line translator won't work!

In C, you can declare variables, in ~~[illegible]~~ any order in a function, but all the memory gets allocated when the function first starts.

```
int main() {
    int x;
    ______
    ______
    ______
    int y;
    ______
    ______
    ______
}
```

all the int declarations happen at beginning when stack frame is made

---

```
int main() {
        ______
        ______
        ______
}
```

what does it mean to translate and execute this line?

Hybrid approaches

~~Python~~ CPython, which is implem at python.org

2 stage:

.pyc

- compiles program to Python bytecode
- interprets bytecode] needs to exist in seperate versions for each kind of machine language

→ Compile can Compile to the same byte code everywhere [you will need to distribute diffent binaries for diffent platforms, but all translate

Python → bytecode

↓

Can make architecture-specific optimizations

Advantage!

Separation of labor ↙

Advantage 2
Compile to bytecode on one computer, and distribute bytecode to anyone, on any processor/OS, because they have their own interpreters.

Oracle Java did this too, but then they switched to:

compiler: $\overset{\text{(Kotlin)}}{\text{Java}}$ $\longrightarrow$ Java bytecode
just-in-time compiler: runs bytecode
↳ compiles pieces of it on-the-fly while running it

BNF - Backus Normal Form

John Backus   ALGOL

```
for ________
begin:
      ______
      ______
      ______

end:
```

denote blocks of code