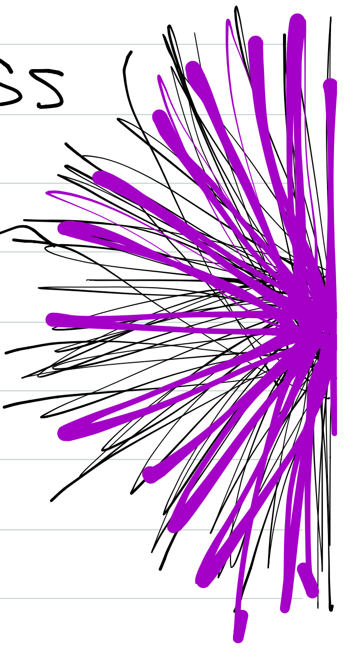
programs/code — not just for machines

* Learning environment (tic-tac-toe machines — reward + punishments)
* Dr Scheme, etc

- Midterm, → during class (

Structure: How we use a computer language to express complicated "computational thoughts" — using "engineering principles to decomp-ose large systems into understandable & manageable parts

Interpretation: (see slide)



Scheme — really simple w/almost no syntax, & not much else (Scheme's programming idiom)

Compiler == more like the translator; the interpreter is the "doer."

(what scheme is supposed to mean in Scheme. Compiling Scheme into Scheme / itself)
↳ (i.e. metacircular evaluation ....)

(what's the model of computation for the language)

Syntax vs. Semantics

Syntax: Say something that's meaningful.

Semantics: say what you meant. (Say "I just want to go to sleep", not "take a sailor to bed.")

(* NB: The read-eval-print loop) → Get cookie when you say "cookie"

---

*Think a/b evolving processes, not the "cash register" model.

---

*Primitive Expressions — "atomic building blocks"

*Means of combinations

*Means of abstraction — Allow compound objects to be named and manipulated almost as if they were primitives.

Simplest binary tree(s) = a leaf. 2 trees together meet @ a node.

Complicated things are inductively defined.

Computational processes... evolve from the interaction of programs (rules) and data (objects to be manipulated).

> 325
325

} Get back the thing itself when you [illegible]

say the word
number

Quotes from Wittgenstein's
Investigations & Augustine's
"Confessions" !