Addendum
(Lecture 13 $\frac{1}{2}$)

Postfix and Infix

Or

Why you should learn to stop worrying and love the stack

Infix – the *fix you know

- You already know infix notation. Easy!
- In algebraic expression with two inputs, we **fix** the operator with**in** the operands:

  5 – 3 = 2

  <operand_1> <operator> <operand_2>
- Things can get ambiguous sometimes:

  5 – 3 x 2 = 4?

  = -1?
- We use order-of-operations rules to resolve these

Postfix – the *fix you need

- Another notational method avoids the ambiguity
- Assume we always use binary operators (limited to two inputs)
- Write the operator after (or **post**) the operands; in other words, write the symbol after the numbers:
  5 3 - = 2
  5 3 2 * - = -1

Postfix – Stacks in the mix

- How does a stack play into this?
- In processing a postfix string, left-to-right
- Basic algorithm idea for a tokenized string:
  - Consider next token
  - If token is a number, push it onto stack
  - If token is an operator …
    - Pop two items (numbers) from stack
    - Perform operation
    - Push result onto stack
  - Loop to consider next token

Postfix – Examples

1 2 + 3 - 4 + 5 6 * *

2 4 8 / * 16 *

10 15 20 + - *

In reverse, infix to postfix: (peel the onion)

((3*4) + 2) / 7 __ __ []