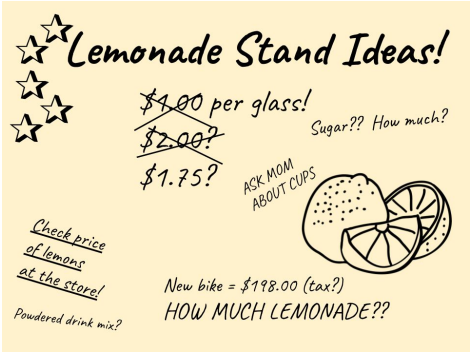
Problem Decomposition

@BootstrapWorld

Problem Decomposition

Sally runs a lemonade stand, which charges $1.75/glass. It costs her $0.30/glass to buy sugar, ice and lemons.

- What do you Notice?
- What do you Wonder?



Problem Decomposition

Complete Word Problems: revenue, cost.

Note: The information you need to write the `cost` *function is provided in the word problem!*

- What is the difference between *revenue* and *profit*?
- How could Sally *increase* her profits?
- What is the *relationship* between profit, cost, and revenue?

Problem Decomposition

- Complete Word Problem: profit.
- When you've completed all three Design Recipes, open Sally's Lemonade Starter File and **Remix/Save a Copy**.
- Type the examples and definitions for all three functions and click Run. Do all your tests pass?

Your teacher may also ask you to complete Sally's Bike

Students browse: www.wescheme.org/openEditor?publicId=vc8K8Xmlin

Problem Decomposition

Turn to Profit - More than one Way! and take a few minutes to reflect on the four function definitions presented.

Problem Decomposition

```
(define (profit g) (- (* 1.75 g) (* 0.30 g)))
(define (profit g) (* (- 1.75 0.30) g))
(define (profit g) (* 1.45 g))
(define (profit g) (- (revenue g) (cost g)))
```

Which of these four `profit` definitions do you think is "best", and why?

Problem Decomposition

Suppose the cost of lemons goes up. Which solution(s) would need to be changed?

What if Sally charges $2/glass? Which solution(s) would need to be changed?

```
(define (profit g) (- (* 1.75 g) (* 0.30 g)))
(define (profit g) (* (- 1.75 0.30) g))
(define (profit g) (* 1.45 g))
(define (profit g) (- (revenue g) (cost g)))
```

Problem Decomposition

`profit` can be *decomposed* into a simpler function that uses `cost` and `revenue`.

Decomposing a problem allows us to solve it in smaller pieces, which are also **easier to test**!

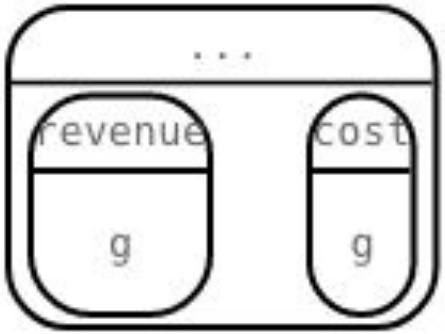
These pieces are **reusable**, resulting in writing **less code**, and **less duplicate code**.

Duplicate code means more places to make mistakes, especially when that code needs to be changed.

Top-Down vs. Bottom-Up

Top-Down and Bottom-Up design are two different strategies for problem decomposition.

Bottom-Up: start with the small, easy relationships like `revenue` and `cost` first. How are they connected with the outer circle? You'll get there eventually, but **we can leave it blank for now (. . .)**. In the Lemonade Stand, you defined `cost` and `revenue` first, and then put them together in `profit`. *This is the same approach as building your Circle of Evaluation inside-out!*



Top-Down vs. Bottom-Up

Top-Down and Bottom-Up design are two different strategies for problem decomposition.

Top-Down: start with the "big picture" and then worry about the details later. We could have started with `profit` as `(- revenue cost)`, and **fill in the details of `revenue` and `cost` later (thus the `...`)**. *This is the same approach as building your Circle of Evaluation outside-in!*

Top-Down vs. Bottom-Up

Jamal's trip requires him to drive 20mi to the airport, fly 2300mi, and then take a bus 6mi to his hotel. His average speed driving to the airport is 40mph, the average speed of an airplane is 575mph, and the average speed of his bus is 15mph. *Aside from time waiting for the plane or bus, how long is Jamal in transit?*

This can be decomposed via Top-Down or Bottom-Up design. What functions would you define to solve this and in what order? Work it out on Top Down or Bottom Up.

Top-Down vs. Bottom-Up

- Whose strategy was Top-Down? How do you know?
- Do you have questions about either of these strategies?
- Which strategy to do you prefer? Why?