Welcome!

We'll get started shortly ...

CS 49 Section

Week 3

Surajit A Bose

Agenda

- Logistics and check-ins
- Review of lecture concepts
  - Decomposition
  - Iterative testing
  - Code style
- Section Problem: Spread beepers

Logistics

How to get hold of me / get help+

- The class forum or section forum, 24 hr turnaround
  - Feel free not only to ask, but also to answer questions there!
- Surajit's office hours:
  - Tuesdays 1p–2p on Zoom
  - By appointment on Zoom or on campus
- Lane's office hours
- Canvas inbox for Lane or Surajit
- Email bosesurajit@fhda.edu, 24 hr turnaround
- Online or in-person tutoring via the STEM center (Room 4213)
- The section GitHub repo has lecture and section slides and solutions

Quick Check-in

Let's quickly go round the room! Pick any one of the following to answer:

- What has been your favorite Karel problem to work on so far?
- What have you found most surprising about the class so far?
- What have you found most difficult about the class so far?

Any Questions?

Flashback to last week …

Functions

- The process of breaking down a problem into smaller, self-contained building blocks is **decomposition**
- These smaller building blocks are **functions**
  - A function is a sequence of steps that achieves a specific outcome
  - Any set of steps that needs to be repeated could be made a function
  - So could any logically self-contained portion of the problem
- How to decompose a problem and write functions:
  - Identify the building blocks
  - Assemble blocks in `main` to solve the big problem
  - Assume the building blocks are done (use the `pass` keyword)
  - Implement each building block!

Lecture Review

What We've Learned This Week

Before we get into our sample problem for today, let's review a bit. This week we've learned:

- Decomposition, aka stepwise refinement or top-down design
  - Breaking down the problem into smaller chunks
- Solving for the general case
- Iterative testing at each step
- Elements of good code style
  - Function names: short, descriptive, **`in_snake_case`**
  - Comments
  - Length of code
  - Indentation levels

Any Questions?

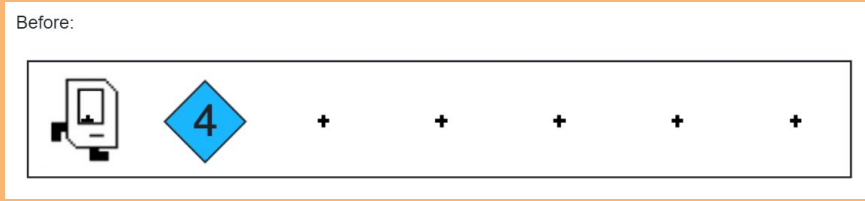
Let's use all that to solve this week's problem!

Section problem: Spread Beepers

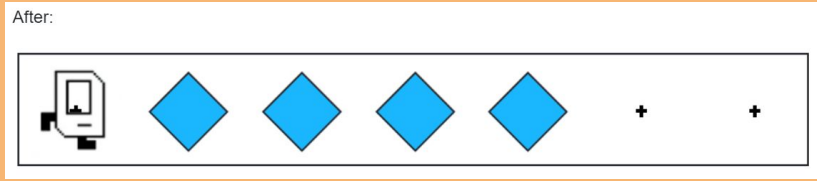
https://codeinplace.stanford.edu/foothill-cs49/ide/a/spreadbeepers

Spread Beepers

- Karel stands directly in front of a pile of a certain number of beepers:



- Those beepers should be spread out across the row:



- The number of beepers is the same before and after they are spread out

Key Details (Assumptions)

- Karel has an infinite number of beepers in its bag!

  This affects what solution you can use.
- There is only one row in the world
- The pile of beepers is in the second column

Questions Before We Begin?

Let's get to work!

Tackling the big picture

Given a problem to solve:

- What is the current situation? : **Preconditions**
  - Note the assumptions in the section handout
- What is the expected outcome? : **Postconditions**
- What steps are needed to get from preconditions to postconditions?
  - Not “how” but “what”; do not worry about the implementation yet
- Do the steps need to be repeated?
  - A certain number of times (`for` loop)
  - As long as a condition is true (`while` loop)

Tackling the building blocks

For each building block: **iterate** and **refine**

- What is the current situation? : **Preconditions**
- What is the expected outcome? : **Postconditions**
- What steps are needed to get from preconditions to postconditions?
  - Implement steps and/or decompose further: **iterate**
  - After each block is built, test it and make improvements: **refine**
- Do the steps need to be repeated?
  - A certain number of times (`for` loop)
  - As long as a condition is true (`while` loop)

Bonus! Extending the problem

Typically*, programs should be written to tackle the general case, not the specific case.

- What if the pile of beepers were anywhere in the row, not just on the second corner? (Assume that the row is still wide enough to spread out the number of beepers in the pile.)
- What if there were multiple rows?

Try it here.

*If we have certain guarantees, then we need not over-engineer the solution to fit the general case: a program to run a task on the last days of January, March, May, July, September, and November need not check if it's a leap year.

Goodbye, Karel!

Next up: Console Programming!