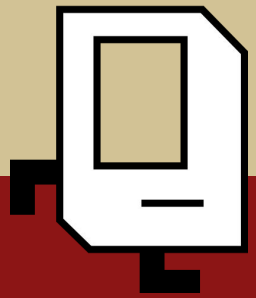
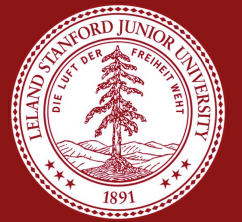


Decomposition with Karel!

Lecture 3

CS106A, Summer 2019
Sarai Gould && Laura Cruz-Albrecht



Announcements

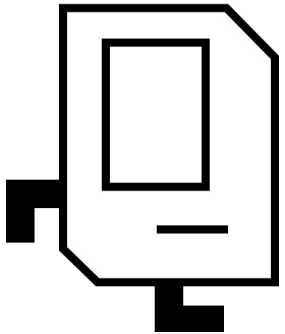
- Section Starts Today!
- LaIR Starts Today!
- If you have midterm conflicts, email both instructors **and** please fill out the form at:
  - http://bit.ly/CS106AMidtermConflicts
- Please email both instructors as soon as possible if you have academic accommodations from the OAE.

Plan for Today

- Assignment 1 - What is It?
- Let's Review!
- More about Methods
- Practice Time!

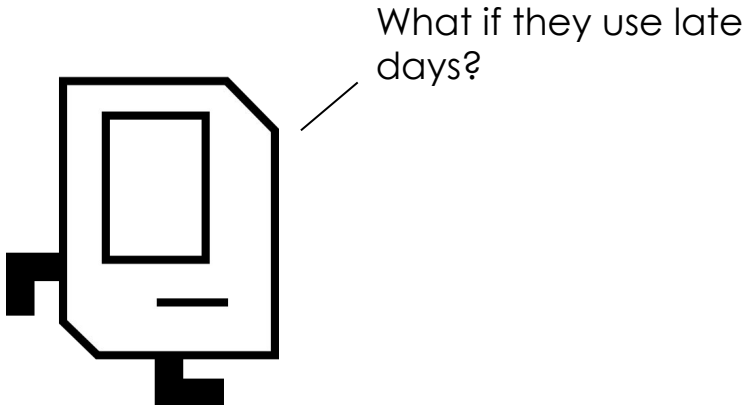
Assignment 1: Karel the Robot

- Assignment 1 Out Today!
  - If you still need help setting up **Eclipse**, please stop by:
    - Sarai and Laura's office hours at **1:30pm - 3:30pm in B02**
    - LaIR at **7pm-11pm in Tressider**
- **Due:** July 3rd at 10am **(30min before lecture!)**



Assignment 1: Karel the Robot

- Assignment 1 Out Today!
  - If you still need help setting up **Eclipse**, please stop by:
    - Sarai and Laura's office hours at **1:30pm - 3:30pm in B02**
    - LaIR at **7pm-11pm in Tressider**
- **Due:** July 3rd at 10am **(30min before lecture!)**



Late Days

- Assignment 1 Out Today!
  - If you still need help setting up **Eclipse**, please stop by:
    - Sarai and Laura's office hours at **1:30pm - 3:30pm in B02**
    - LaIR at **7pm-11pm in Tressider**
- **Due:** July 3rd at 10am **(30min before lecture!)**

- One late day is a ***24 hour extension*.**
  - **Instead of being due July 3rd at 10am, it's now due July 4th at 10am!**
- You have ***three* late days** for the whole quarter!

Honor Code

Honor Code

- Do **not look at assignment solutions that are not your own**.
  - Online or another student's.

Honor Code

- Do **not look at assignment solutions that are not your own**.
  - Online or another student's.
- Do **not share solutions with other students**.

Honor Code

- Do **not look at assignment solutions that are not your own**.
  - Online or another student's.
- Do **not share solutions with other students**.
- If you discuss strategies (NOT SOLUTIONS), **you must indicate assistance you have received**.
  - You do not need to do this if the person is a CS106 staff member.

Honor Code

- Do **not look at assignment solutions that are not your own**.
  - Online or another student's.
- Do **not share solutions with other students**.
- If you discuss strategies (NOT SOLUTIONS), **you must indicate assistance you have received**.
  - You do not need to do this if the person is a CS106 staff member.
- You **can only reuse work in certain, limited situations**.

Honor Code

- Do **not look at assignment solutions that are not your own**.
  - Online or another student's.
- Do **not share solutions with other students**.
- If you discuss strategies (NOT SOLUTIONS), **you must indicate assistance you have received**.
  - You do not need to do this if the person is a CS106 staff member.
- You **can only reuse work in certain, limited situations**.
- If you have questions about honor code, please ask the instructors.

Honor Code

There is an automated program that will compare all of your programs to past, present, and known solutions to the assignments.

We will catch you if you violate the honor code.

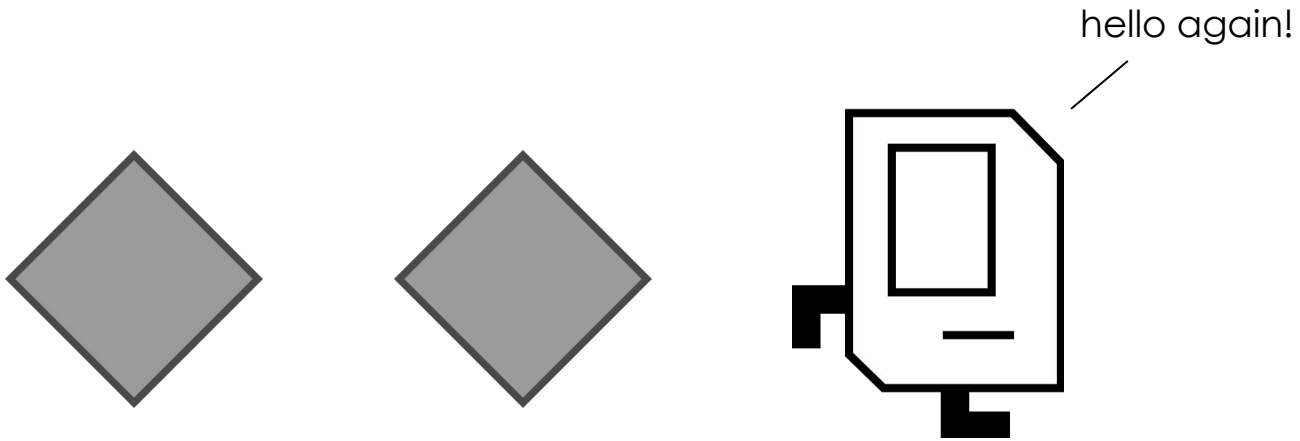
Honor Code

There is an automated program that will compare all of your programs to past, present, and known solutions to the assignments.

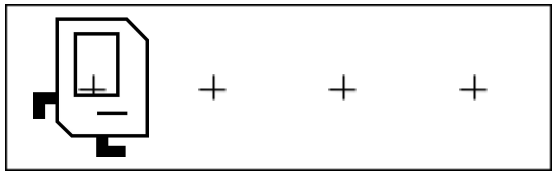
We will catch you if you violate the honor code.

If you violate the honor code, you will fail this course.

Back to Karel!



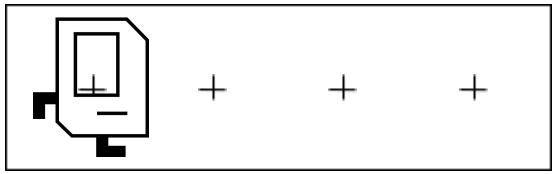
Review: A Method



A **method** is a set of new instructions we've created!

```
/* Comment describing method */
private void nameOfMethod(){

     // command 1
     // command 2

}
```

Review: For Loops

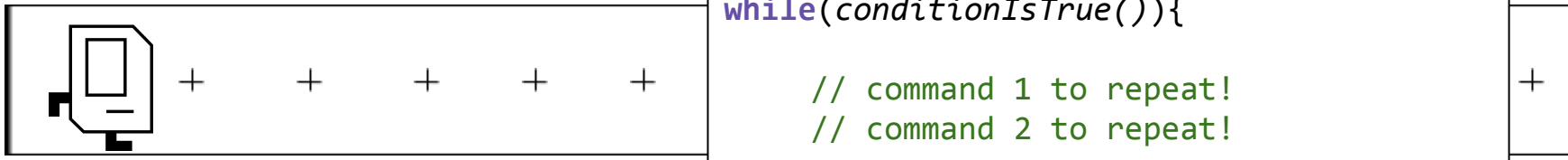


For Loops repeat something a *specific number of times!*

```
for(int i = 0; i < num_times; i++){

    // command 1 to repeat!
    // command 2 to repeat!

}

// code out here is NOT repeated!
```

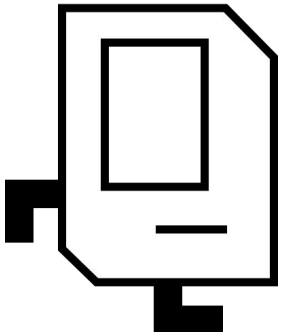
Review: While Loops

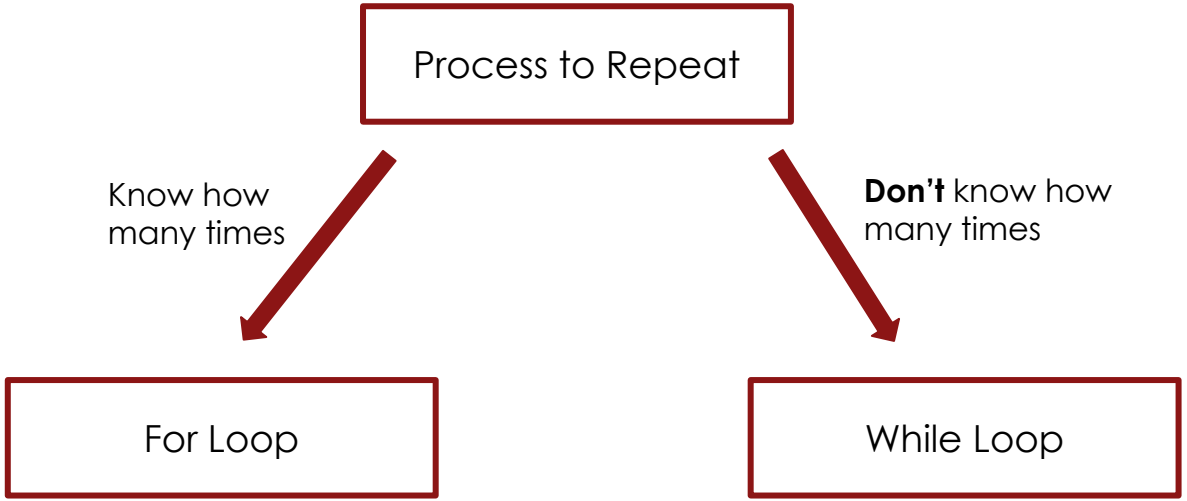
`While Loops` repeat *until a condition is no longer met!*



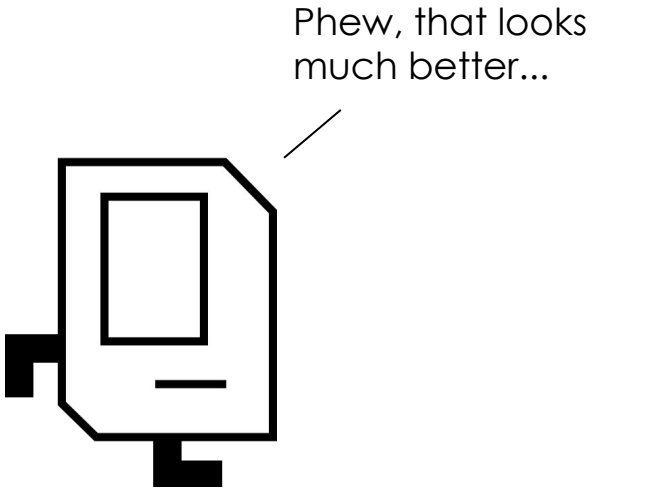
```
while(conditionIsTrue()){

    // command 1 to repeat!
    // command 2 to repeat!

}

// the condition isn't true anymore,
// so our code exited the while loop.
// code out here is NOT repeated!
```

While Loop or For Loop?





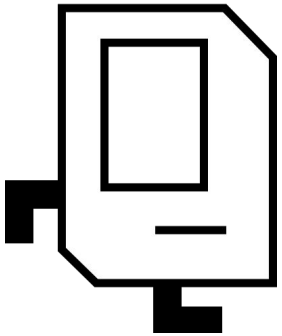
Review: If Statements



`If statements` check a condition *once*!

```
if(conditionIsTrue()){

     // command 1!
     // command 2!

}

// we have left the if statement.
// code out here happens no matter what!
```

Review: If-Else



The **`else`** code only runs if the condition is *not* true!

```
if(conditionIsTrue()){

     // command 1 if conditionIsTrue()!
     // command 2 if conditionIsTrue()!

}  else {

     // command 1 if conditionIsFalse()!
     // command 2 if conditionIsFalse()!

}
// code out here happens no matter what!
```

What Conditions Can Karel Check?

Test	*Opposite*	*What it checks*
`frontIsClear()`	`frontIsBlocked()`	Is there a wall in front of Karel?
`leftIsClear()`	`leftIsBlocked()`	Is there a wall to Karel's left?
`rightIsClear()`	`rightIsBlocked()`	Is there a wall to Karel's right?
`beepersPresent()`	`noBeepersPresent()`	Are there beepers on this corner?
`beepersInBag()`	`noBeepersInBag()`	Any there beepers in Karel's bag?
`facingNorth()`	`notFacingNorth()`	Is Karel facing north?
`facingEast()`	`notFacingEast()`	Is Karel facing east?
`facingSouth()`	`notFacingSouth()`	Is Karel facing south?
`facingWest()`	`notFacingWest()`	Is Karel facing west?

Combining Conditions: && and | |

|| means "or". **One or both conditions** can be true!

```
if(condAIsTrue() || condBIsTrue()){

    // This code will run if one or
    // both of the conditions is true!

}
```

Combining Conditions: && and | |

&& means "and". **Both conditions** must be true!

```
if(condAIsTrue() && condBIsTrue()){

    // This code will only run if both
    // of the conditions are true!

}
```

Combining Conditions: && and | |

|| means "or". **One or both conditions** can be true!

```
if(condAIsTrue() || condBIsTrue()){

    // This code will run if one or
    // both of the conditions is true!

}
```

&& means "and". **Both conditions** must be true!

```
if(condAIsTrue() && condBIsTrue()){

    // This code will only run if both
    // of the conditions are true!

}
```

Decomposition: What Is It?

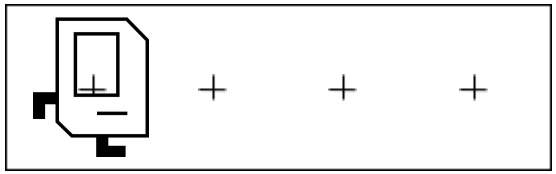
Decomposition is the process of **breaking something down into smaller pieces**.

Decomposition: What Is It?

Decomposition is the process of **breaking something down into smaller pieces**. We create well-defined steps to describe what we want to do.

In programming, we often use short, descriptively-named **methods** to decompose our code into reasonable pieces.

A Method



A **method** is a set of new instructions we've created!

```
/* Comment describing method */
private void nameOfMethod(){

     // command 1
     // command 2

}
```

Decomposition: What Is It?

Essentially, we're breaking our code down into **different, small sets of instructions for the computer** (or Karel) **to follow!**

Each method should describe one task. If you find yourself creating instructions for multiple tasks in one method you should probably create another method!

Example of Decomposing

Example of Decomposing

NOT Decomposed:

Pick up toothbrush
Put toothpaste on toothbrush
Put toothbrush in mouth
Move toothbrush back and forth
Take toothbrush out of mouth
Wash toothbrush
Put down toothbrush
Pick up hair brush
Run hair brush through hair
Put down hair brush
Open cupboard
Take out bowl
Take out cereal
Close cupboard
Pick up cereal
Pour cereal into bowl
Open fridge
Take out milk
Pour milk into bowl

Decomposed:

Brush Teeth

Brush Hair

Eat Breakfast

Example of Decomposing

NOT Decomposed:

Pick up toothbrush
Put toothpaste on toothbrush
Put toothbrush in mouth
Move toothbrush back and forth
Take toothbrush out of mouth
Wash toothbrush
Put down toothbrush
Pick up hair brush
Run hair brush through hair
Put down hair brush
Open cupboard
Take out bowl
Take out cereal
Close cupboard
Pick up cereal
Pour cereal into bowl
Open fridge
Take out milk
Pour milk into bowl

Decomposed:

Brush Teeth

Brush Hair

Eat Breakfast

We can define these later on!

Example of Decomposing

Note:

It's okay to figure out what the big steps are first and then define them later on! This can help you create a well-organized program and not get caught up on tiny details.

Decomposed:

- Brush Teeth
- Brush Hair
- Eat Breakfast

We can define these later on!

Quick Practice

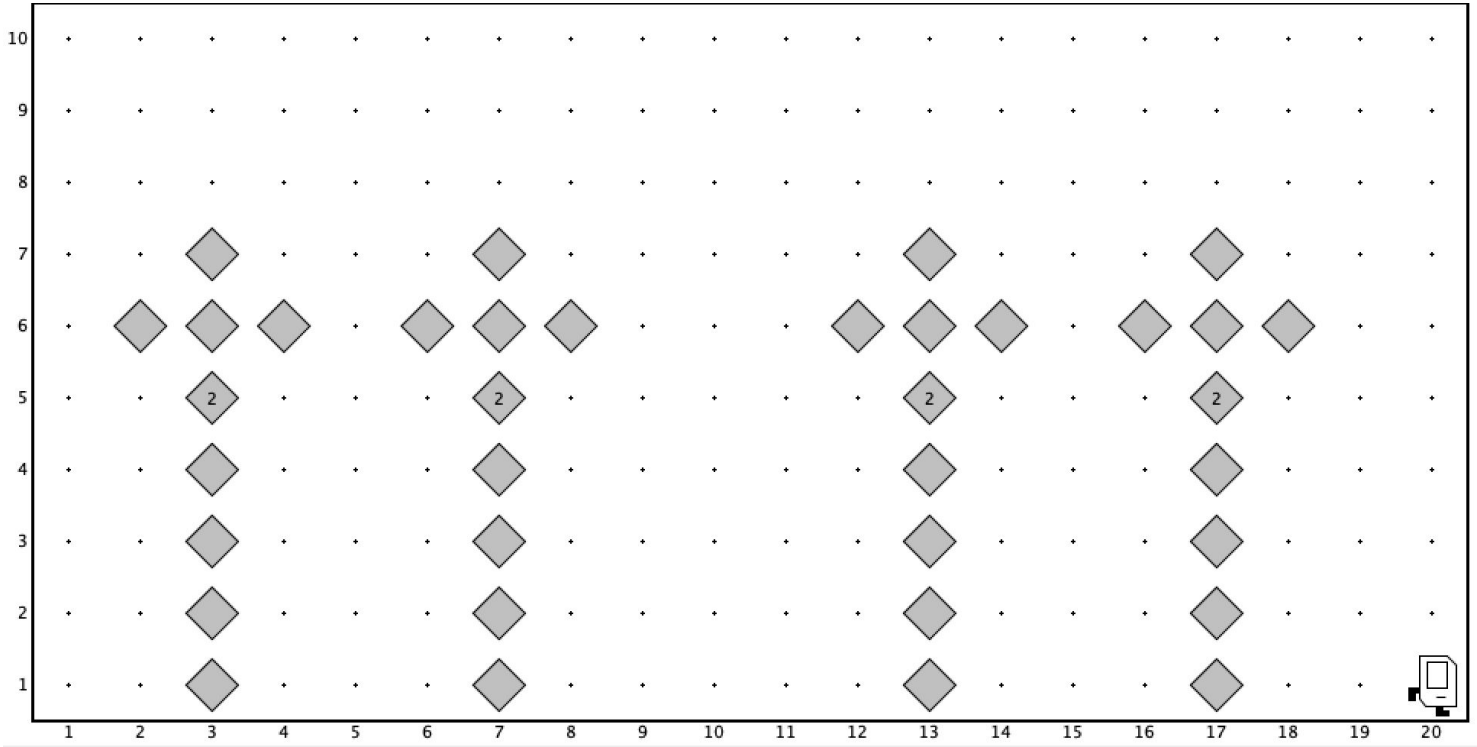
Pick one process you do every day.

First, think of how you would describe it *without using decomposition*. Next, think of how you would describe it *using decomposition*.

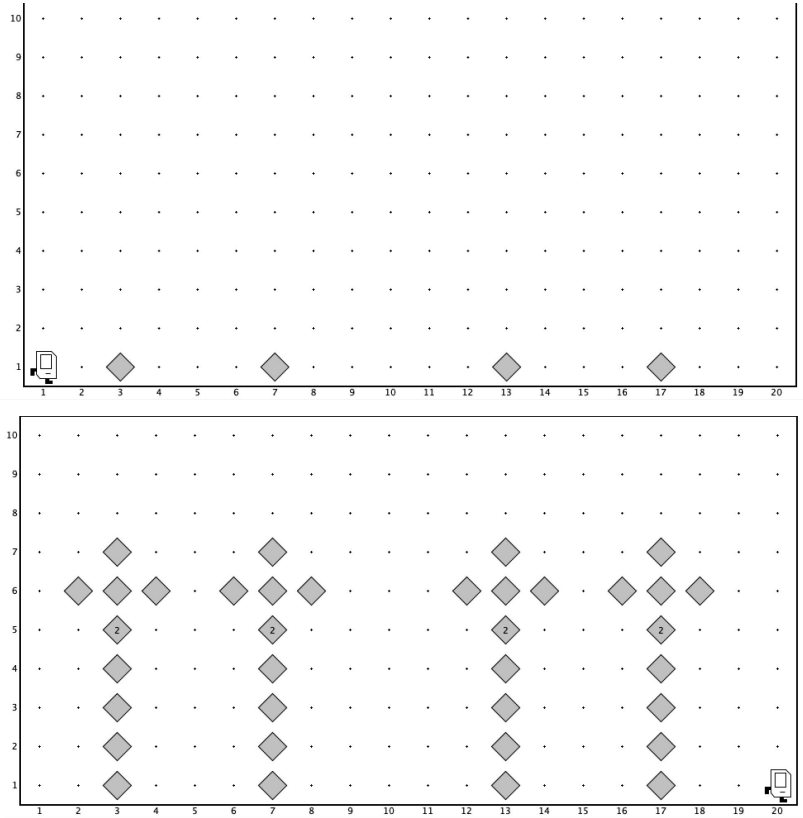
Share with someone sitting next to you.

Let's Look at a Code Example!

Drawing Palm Trees!



Drawing Palm Trees!

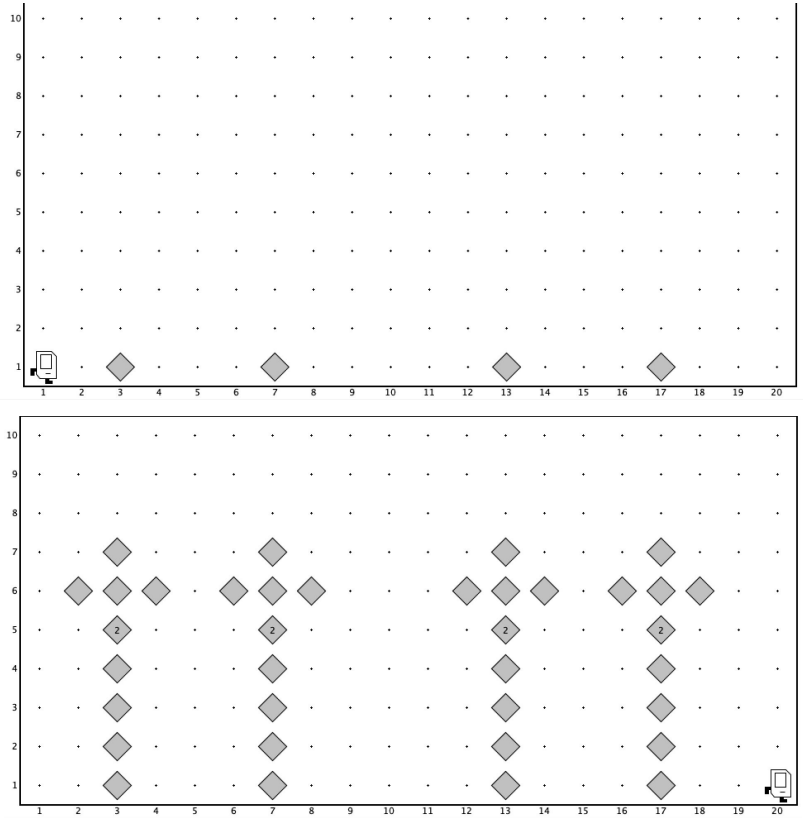


What's the Task?

Whenever Karel runs into a beeper, Karel will draw a Palm Tree!

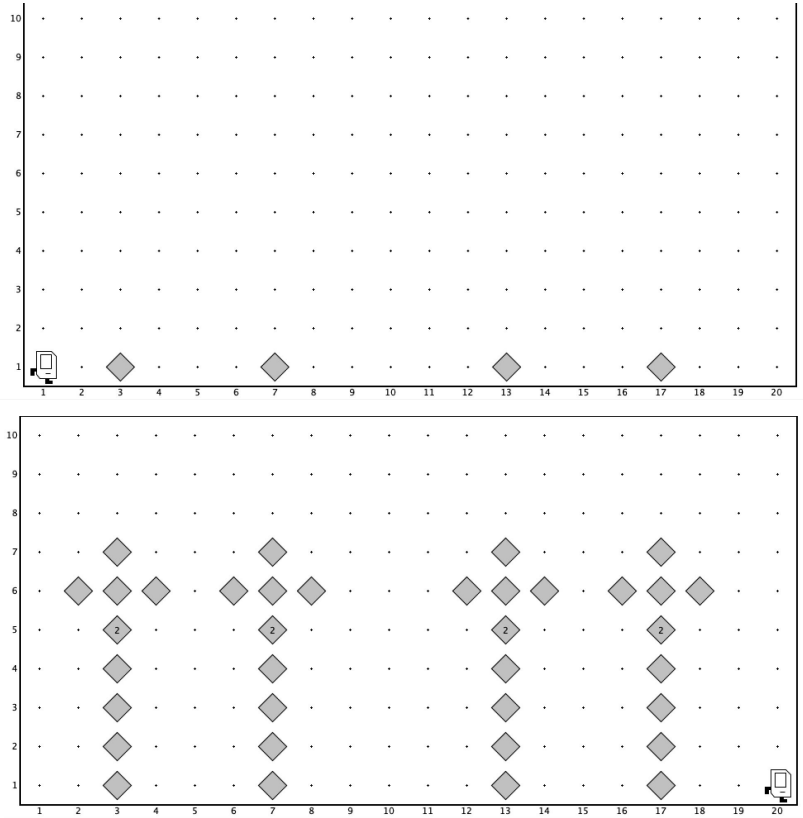
- Palm trees are **6 beepers tall**.
- Palm trees have **4 leaves**, each **1 beeper long**.
- We don't know the length of the world.
- If there is a beeper, there will be room to draw a palm tree at that location.

Drawing Palm Trees!



What's the Pseudocode?

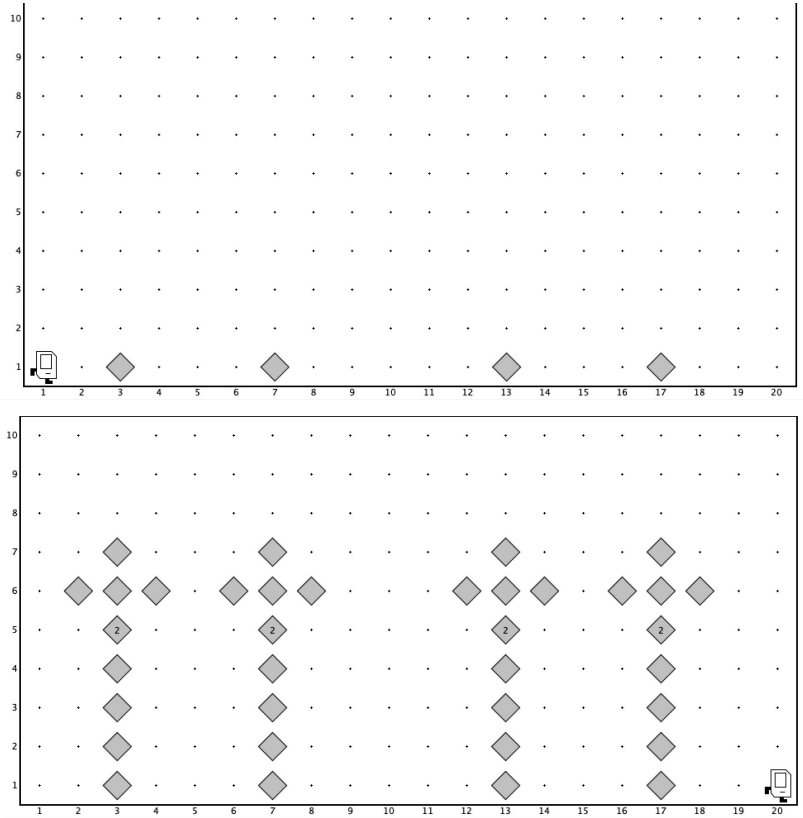
Drawing Palm Trees!



My Pseudocode:

```
while nothing is in front of Karel:
    move
    if Karel sees a beeper:
        draw a tree!
```

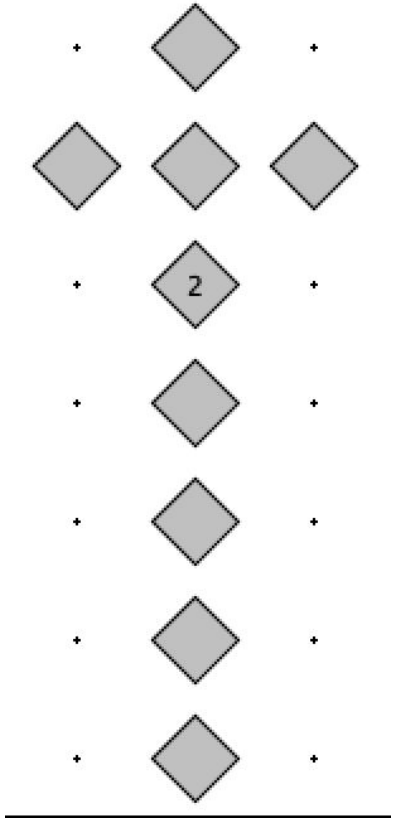
Drawing Palm Trees!



My Pseudocode:

while nothing is in front of Karel:
    move
    if Karel sees a beeper:
        draw a tree!

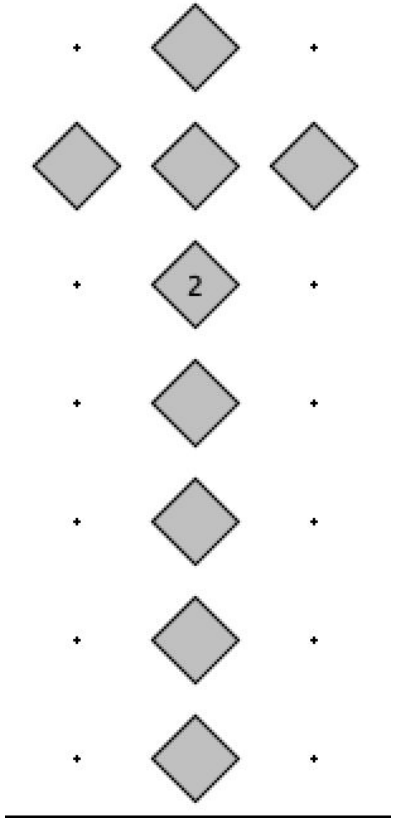
Drawing Palm Trees!



My Pseudocode for Drawing a Tree:

draw a trunk
draw leaves
descend trunk

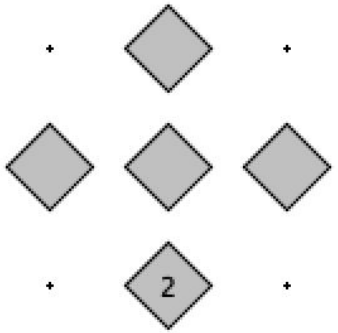
Drawing Palm Trees!



My Pseudocode for Drawing a Tree:

draw a trunk
draw leaves
descend trunk

Drawing Palm Trees!



My Pseudocode for Drawing Leaves:

repeat 4 times:
    draw a leaf
    move back to trunk

Drawing Palm Trees!

My Starting Pseudocode:

```
while nothing is in front of Karel:
        move
        if Karel sees a beeper:
                draw a tree!
```

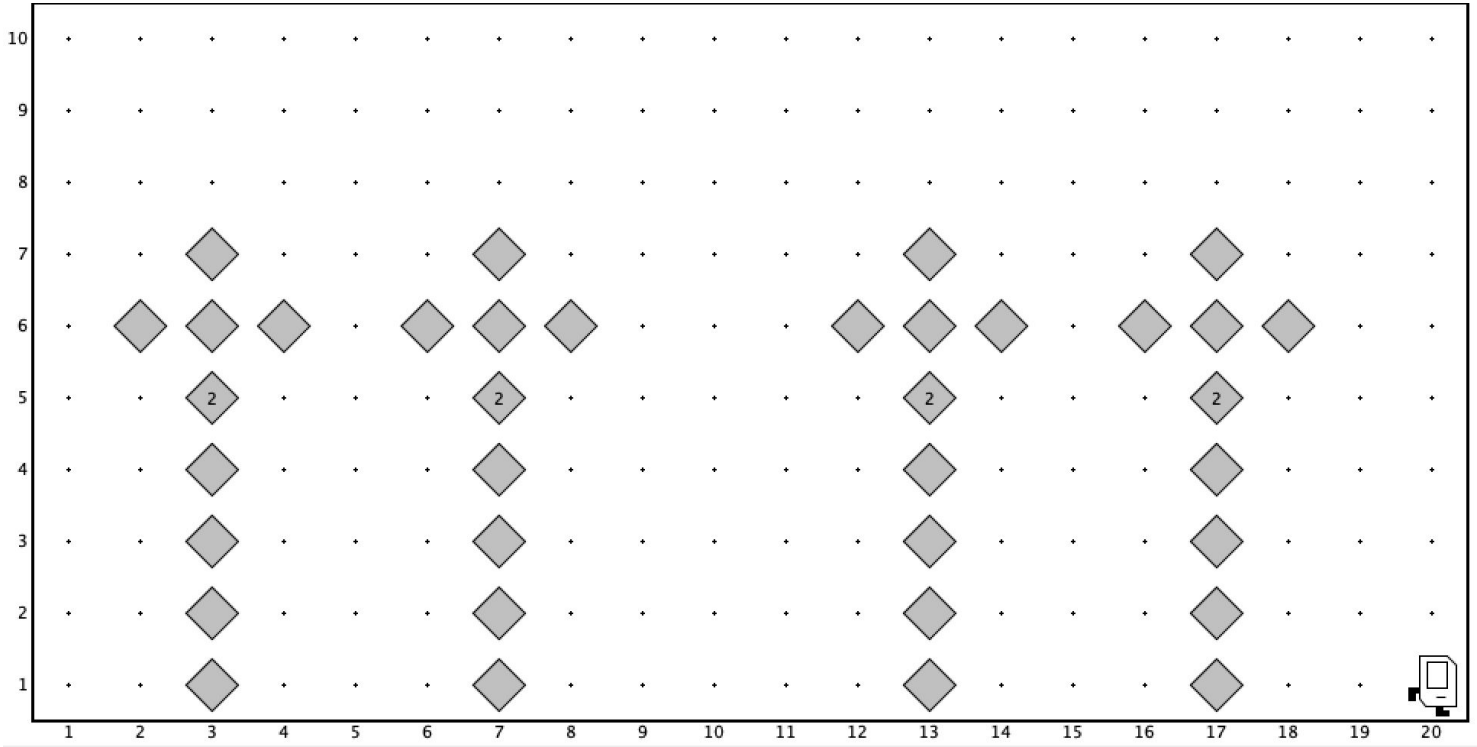
My Pseudocode for Drawing a Tree:

```
draw a trunk
draw leaves
descend trunkd
```

My Pseudocode for Drawing Leaves:

```
repeat 4 times:
        draw a leaf
        move back to trunk
```

Drawing Palm Trees!



Let’s Code It!

Pre and Post Conditions

Every new set of instructions, or new **method**, usually:

- Assumes something is true before it begins.
- Promises something is true at the end.

This is the **Pre-condition** and **Post-condition** of your **method.**

Pre and Post Conditions

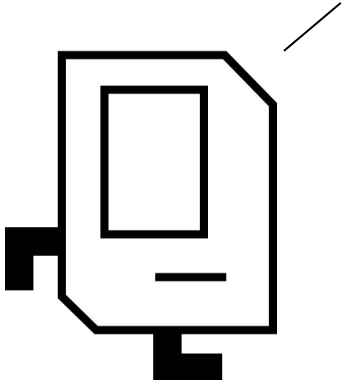
Example of a comment explaining the Pre and Post conditions for our `drawATree()` method!

```
/* Karel will draw a palm tree!
* Pre: Karel is facing North.
* Post: Karel has drawn a tree and is
* facing South.
*/
private void drawATree(){
     drawTrunk();
     drawLeaves();
     descendTree();

}
```

More Isn't Always Better

If it doesn't work, sometimes it makes sense to **start over.**

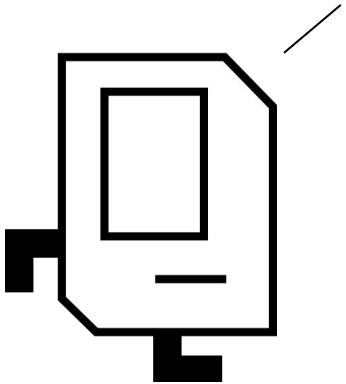
You should **remove destructive code, rather than always adding corrective code.**



Infinite Loops

An infinite loop is a loop that **never ends**.

How can that happen?



What's the Problem?

```
import  stanford.karel.*;

public class InfiniteLoop extends SuperKarel {

     public void run(){
          findOpening();
          move();
     }

     private void findOpening(){
          while(frontIsBlocked()){
               turnRight();
          }
     }
}
```

What's the Problem?

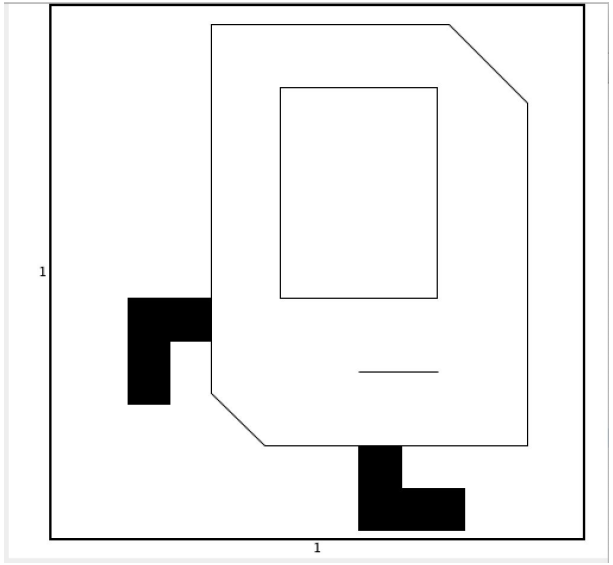
```
import  stanford.karel.*;

public class InfiniteLoop extends SuperKarel {

     public void run(){
          findOpening();
          move();
     }

     private void findOpening(){
          while(frontIsBlocked()){
               turnRight();
          }
     }
}
```

Do you notice anything about this code block?

What's the Problem?

```
import  stanford.karel.*;

public class InfiniteLoop extends SuperKarel {

     public void run(){
          findOpening();
          move();
     }

     private void findOpening(){
          while(frontIsBlocked()){
               turnRight();
          }
     }
}
```

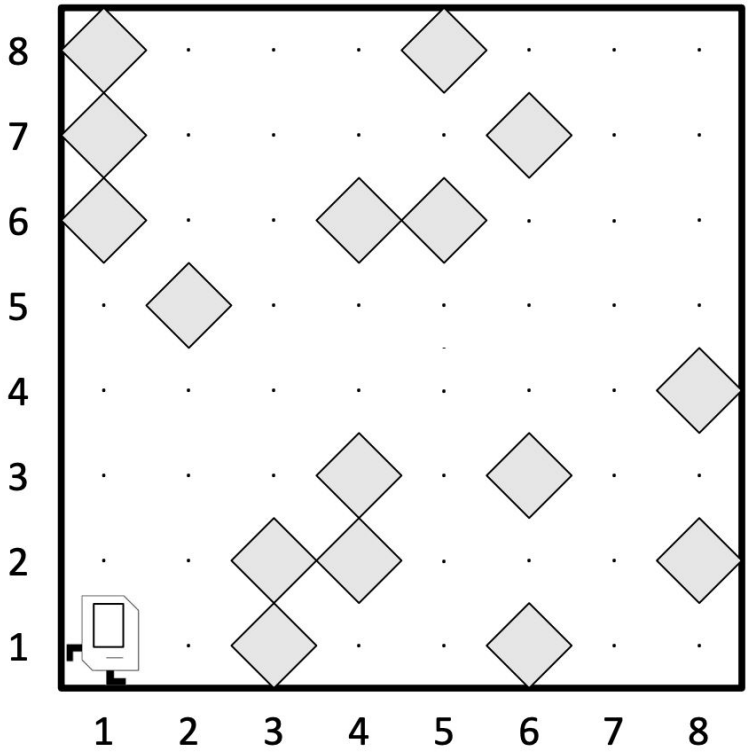
What if THIS 1x1 corner is Karel's world?



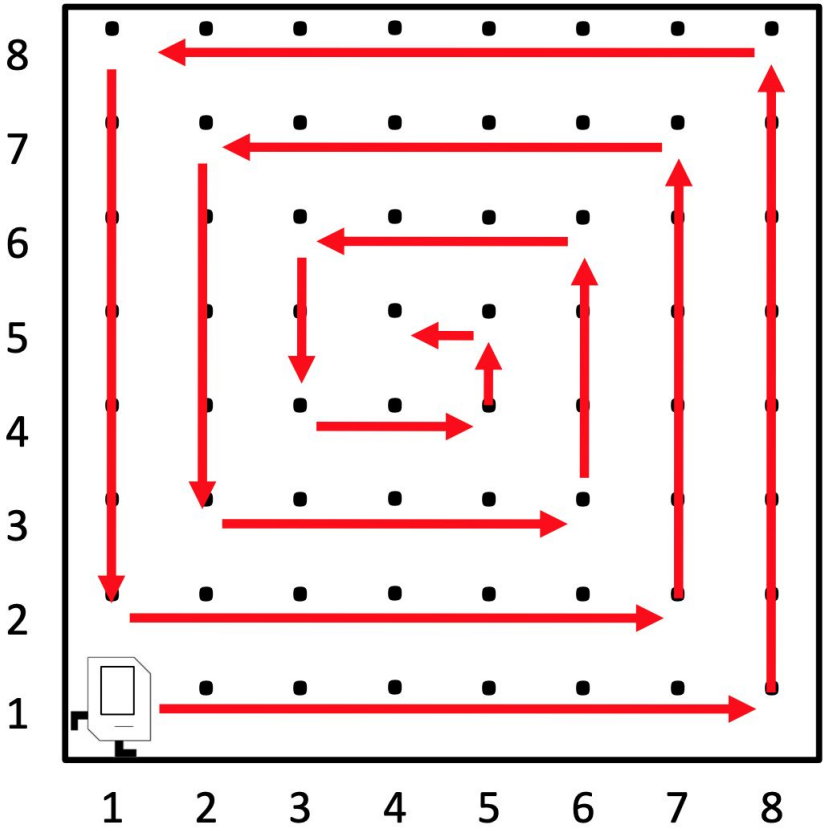
Practice: Roomba Practice

Create a Karel program that will clear the whole world of beepers!

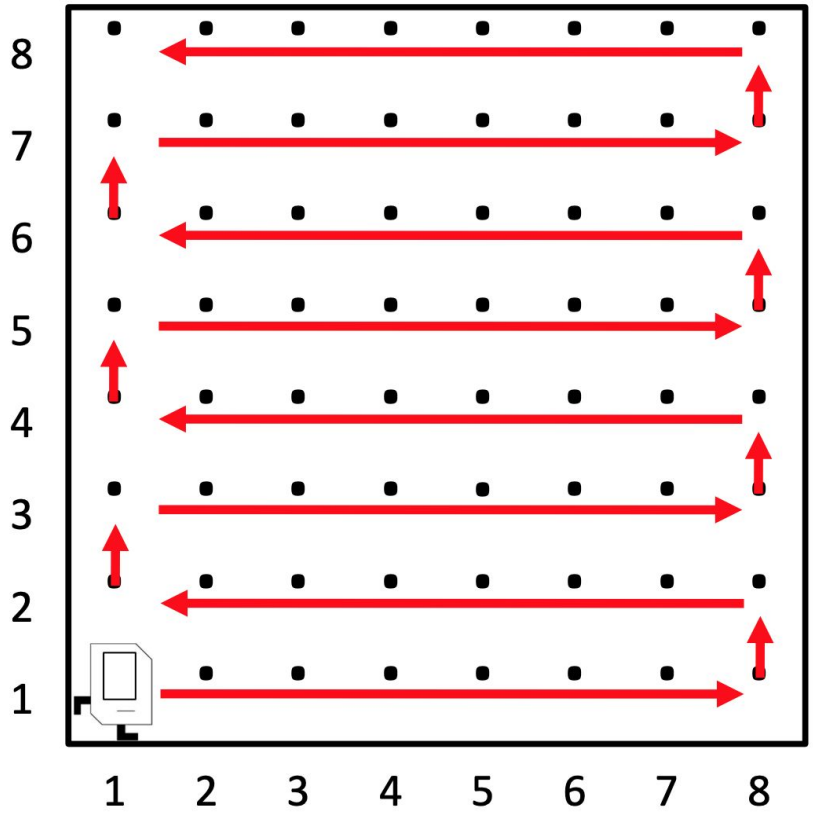
- Karel will always start at (1,1).
- It doesn't matter where Karel ends.
- There must be no beepers remaining at the end of the program.



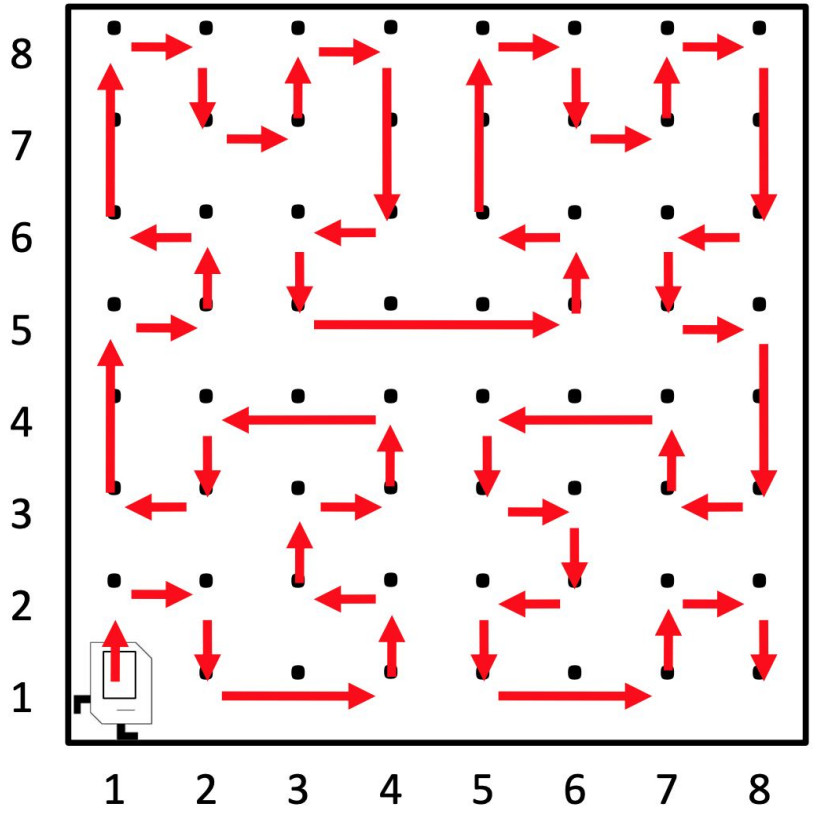
Algorithm 1



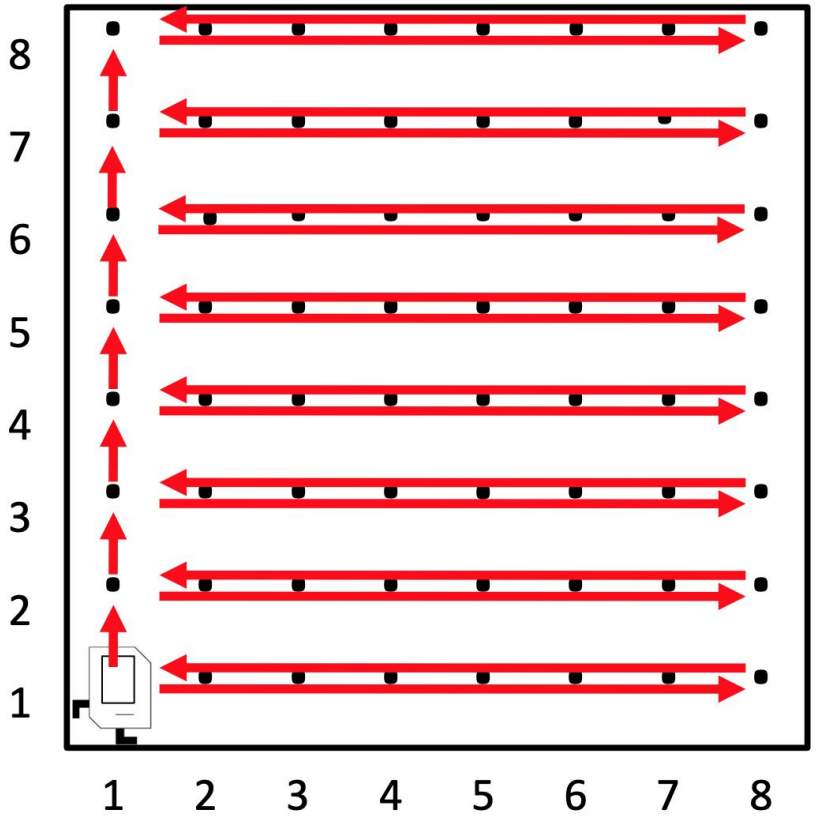
Algorithm 2



Algorithm 3



Algorithm 4

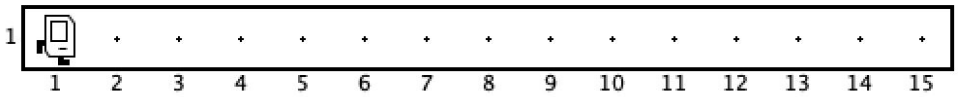


Practice: Counting By Beepers

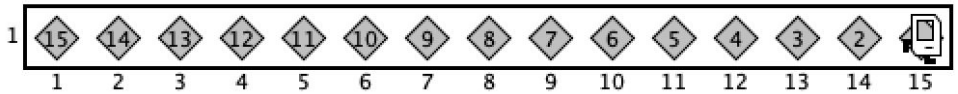
Create a Karel program in which Karel will place an decreasing number of beepers in each space.

- Karel always starts at (1,1).
- The world is always 1 tall.
- The world always starts with no beepers.
- We don't know how long the world will be.

Before:



After:



Plan for Today

- Assignment 1 - What is It?
- Let's Review!
- More about Methods
- Practice Time!

Homework:

- **Assignment 1** goes out today! Due July 3rd at 10am.
- **If you have midterm conflicts, email instructors and fill out this form:** http://bit.ly/CS106AMidtermConflicts
- **Email both instructors if you have academic conflicts from the OAE.**