Control Flow with Karel

Table of contents

This lesson contains a bunch of Karel problems with solutions, tagged by the concepts they cover. The problems are vaguely ordered from easiest to hardest, and the types of problems are described below. We recommend you use this as a resource rather than running it through front to back -- pick which concepts you need the most practice with, find a problem (or multiple!) that covers those concepts and looks fun, and have a good time! Whether you do these problems yourself or go straight to reading the solutions, we hope these will be helpful. Happy coding!

Control flow warmups

These problems are super short and should be a nice introduction to the concepts.

Larger problem, with milestones: spring flowers

This problem is somewhat big (similar to assignment problems), but don't worry -- we've split it into smaller, very manageable milestones, which are tagged by the concepts they cover. If you're having trouble knowing how and when to decompose a problem into smaller subproblems, reading through the milestones might be helpful! If you'd prefer to do this problem without following our milestones, skip to Milestone 4 and have a go!

- Spring flowers, Milestone 1: bloom (basic Karel commands, optional for loop)
- Spring flowers, Milestone 2a: move to wall (while loops)
- Spring flowers, Milestone 2b: climb stem (while loops)
- Spring flowers, Milestone 3: bloom flower (decomposition)
- Spring flowers, Milestone 4: bloom all the flowers!! (for loops, decomposition)

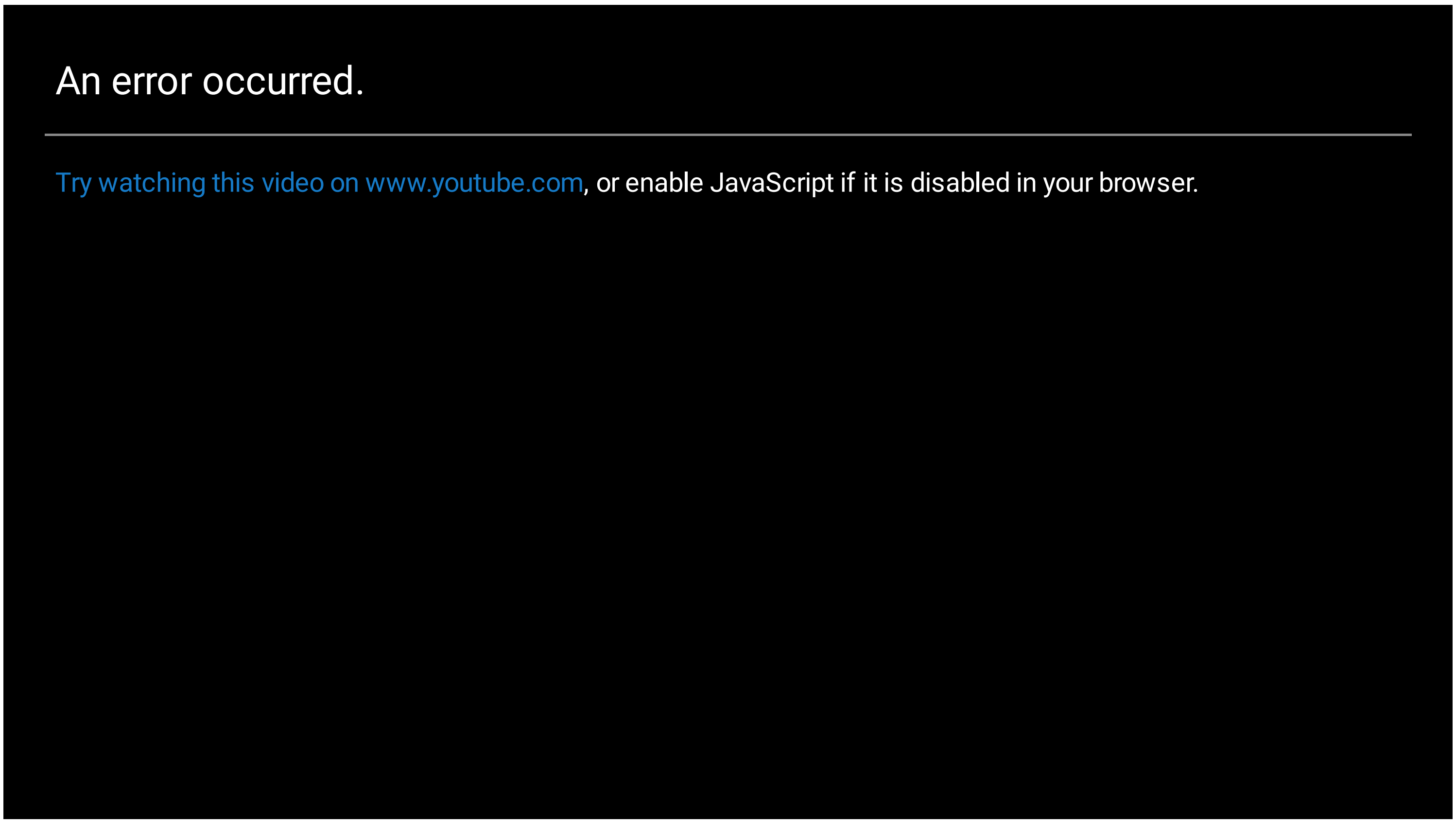
More problems

These problems have predefined worlds and tasks and are vaguely ordered from easiest to hardest. Here, we haven't written the milestones out for you, but decomposition will be very helpful!

- Three slots (for loops, fencepost)
- 10s across the board (for loops, while loops, fencepost)
- Maximum 5 (for loops, if/else)
- Invert (while loops, if/else)
- Five corridors (for loops, while loops, fencepost, if/else)
- Upstairs, downstairs (for/while loops)
- Zig zag (while loops, optional if/else)
- Labyrinth (while loops, if/else)
- Treasure hunt, part 1 (while loops)
- Treasure hunt, part 2 (while loops, if/else)

Tips for using the Karel editor & troubleshooting

Here's a video tutorial for how to use Karel on Ed:



An error occurred.

Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.

Here are some tips and tricks for using the Karel editor (if you have a Karel editor related question that isn't answered here, please post it in the Discussion and we'll update this page!)

- If you'd like to **make your own world for Karel**, change the last line of the file to `run_karel_program()` if it isn't already, and use the Karel editor pane to change the world dimensions. To teleport Karel to a different spot, place and remove beepers, or change Karel's direction, right click on the world and select what you'd like to do, and then click a spot in the world to execute that action.
- If for some reason the world specified in the **problem doesn't load automatically**, pressing Run and then Reset with no code should do the trick.

For loop warmups

Recall that, conceptually, for loops are used when you need to *do something a fixed number of times*. The following exercises are some straightforward practice with `for` loops.

Place 10 beepers

Place 10 beepers on the spot Karel is currently standing on.

Move 5

Move forward 5 times.

Backflip

Get Karel to do a cool backflip by turning left 4 times.

Square

Make Karel place beepers in a square (4 beepers total) and end in the same position Karel starts in.

4 in a row

Put 4 beepers down in a row, starting with Karel's current position. In other words, if Karel starts at row 1 column 1 facing East, place beepers in row 1 column 1, row 1 column 2, and row 1 column 3. **This exercise tests your understanding of the fencepost problem from lecture!**

While loop warmups

Recall that `while` loops are used when you want to *do something until a certain condition is met.* The following problems are some exercises you can do for some straightforward `while` loop practice.

Move to wall

Move Karel forward until you run into a wall (don't walk through the wall!).

Clean spot

Write code which will "clean" the spot Karel is standing on by picking up beepers until there aren't any left (you can't `pick_beeper()` if there aren't any!).

Turn to wall

Turn left until Karel is facing a wall.

Fill bottom row

Fill the entire bottom row of the world with beepers, making sure not to forget to put a beeper on the spots Karel starts / ends on. **This tests your understanding of the fencepost problem from lecture.**

If/else warmups

We can use `if` and `else` statements to control what happens based on the condition we check for. The following exercises test your understanding of conditional logic.

Invert spot

Invert the spot Karel is currently standing on -- if there is a beeper present, pick it up; if there isn't a beeper present, put one down. (This may be a convenient helper function for the "Invert" practice problem later in this lesson!)

We've provided you two worlds on which to test your code. The `InvertBeeper` world (1x1, with a beeper) will load by default, but you can toggle to the `InvertNoBeeper` (1x1 without a beeper) world by changing the very last line in the file from `run_karel_program('InvertBeeper.w')` to `run_karel_program('InvertNoBeeper.w')` (and vice versa). You'll need to press Run to see the world change.

Move if clear

Write code to move forward once and put a beeper down if Karel isn't facing a wall. If Karel is facing a wall, don't move and just put a beeper down.

One solution to this uses an `if` statement and an `else` statement. One solution to this just uses an `if` statement. Can you think of both?

We've provided you two worlds on which to test your code. The `FrontClear` world (in which Karel isn't facing a wall) will load by default, but you can toggle to the `FrontNotClear` world by changing the very last line in the file from `run_karel_program('FrontClear.w')` to `run_karel_program('FrontNotClear.w')` (and vice versa).

Note: if you run `put_beeper()` and it says “No beepers in bag”, your world file is likely out of date -- we've since fixed this! Save a copy of your code somewhere safe and then click the ... > Reset to Scaffold button in the top right to update everything automatically.

Conditional turn

Get Karel to turn based on whether or not a beeper is present -- if there is a beeper in the current spot, turn left; if there isn't a beeper in the current spot, turn right.

We've provided you two worlds on which to test your code. The `1x1Beeper` world will load by default, but you can toggle to the `1x1NoBeeper` world by changing the very last line in the file from `run_karel_program('1x1Beeper.w')` to `run_karel_program('1x1NoBeeper.w')` (and vice versa).

Turn signal

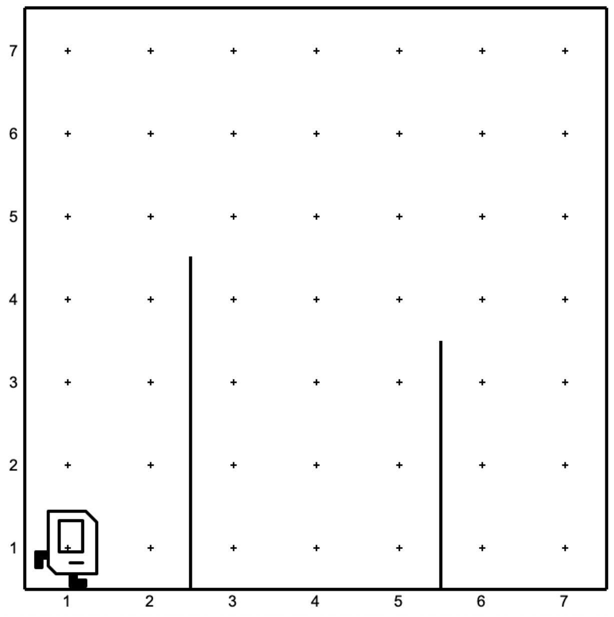
If Karel is facing a wall, put a beeper down, turn left, and move forward. If not, do nothing.

We've provided you two worlds on which to test your code. The `FrontBlocked` world (in which Karel is facing a wall) will load by default, but you can toggle to the `FrontClear` world by changing the very last line in the file from `run_karel_program('FrontBlocked.w')` to `run_karel_program('FrontClear.w')` (and vice versa).

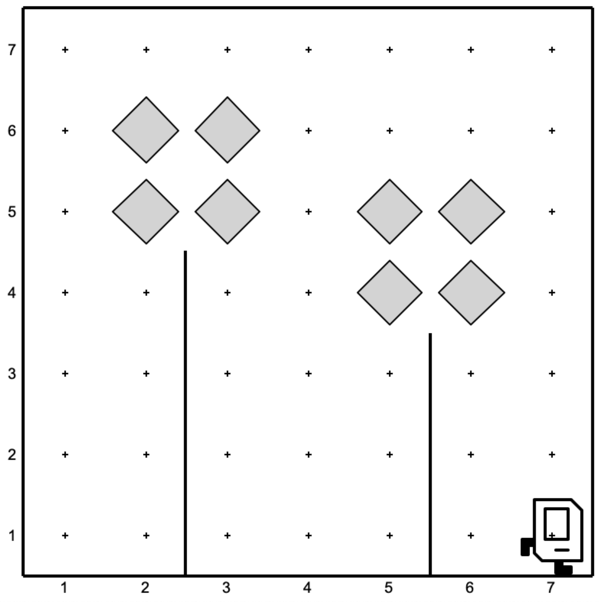
Note: if you run `put_beeper()` and it says “No beepers in bag”, your world file is likely out of date -- we've since fixed this! Save a copy of your code somewhere safe and then click the ... > Reset to Scaffold button in the top right to update everything automatically.

Spring flowers!

For this problem, Karel will start in a world with exactly 2 flower stems, like so:



Since it's springtime, Karel will make the "flowers" bloom! Each flower is comprised of four beepers and goes directly on top of each stem, so that the end result for this world looks like this (note that Karel should end up in the bottom right corner):



Some important notes:

- There will always be exactly 2 stems, but
- they may not always be the same distance apart, though they will be spaced so that you will always be able to bloom flowers without the blooms overlapping.
- The stems may be of any height, but they will be short enough that you can bloom an entire

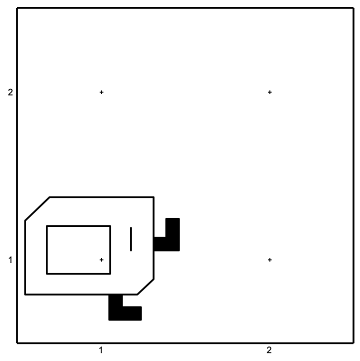
flower without walking off of the top edge of the world.

- Karel will always start in the bottom left corner of the world, and Karel should always end in the bottom right corner of the world.

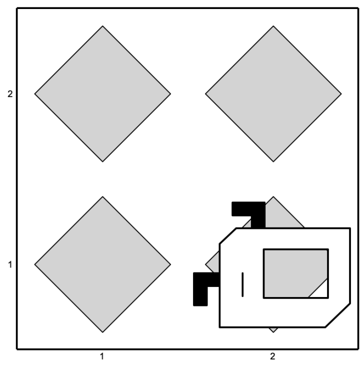
We've split this problem into milestones that we think subdivide the problem into reasonable subproblems. If you'd prefer not to follow our milestones, feel free to skip to the last milestone, Milestone 4, and write the solution from scratch!

Spring flowers, Milestone 1: bloom (basic Karel commands, optional for loop)

Write a program which will bloom a flower by placing 4 beepers in a square, assuming Karel starts in the bottom left corner of the square facing North. For example, Karel will start like this:



And should end like this:



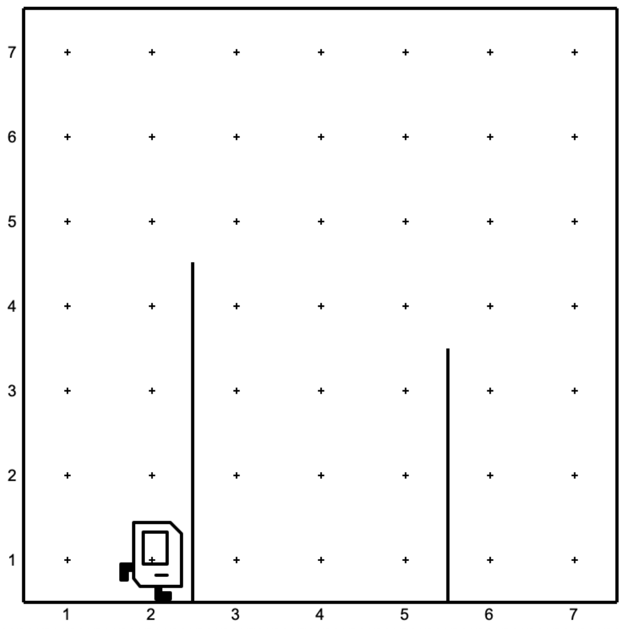
Note: if you run `put_beeper()` and it says "No beepers in bag", your world file is likely out of date -- we've since fixed this bug! Save a copy of your code somewhere safe and then click the ... > Reset to Scaffold button in the top right to update everything automatically.

Spring flowers, Milestone 2a: move to wall (while loops)

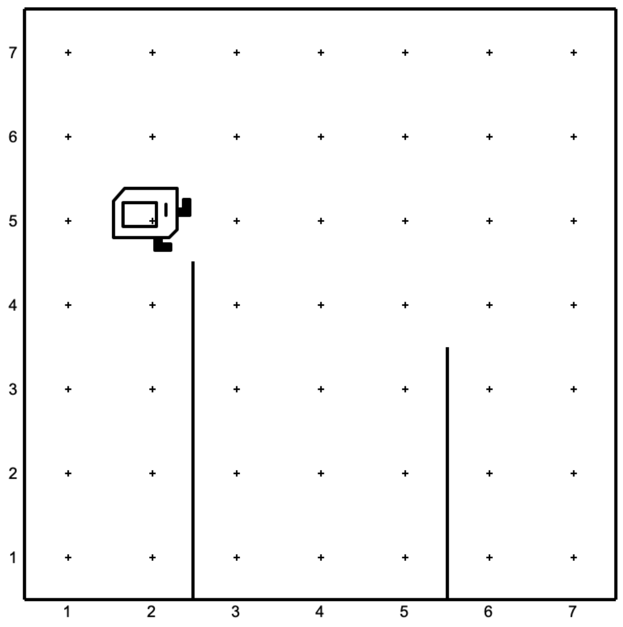
Write a program which will move Karel forward until she reaches a wall. This function will come in handy later.

Spring flowers, Milestone 2b: climb stem (while loops)

Assuming Karel starts facing East at the base of a stem, like so:



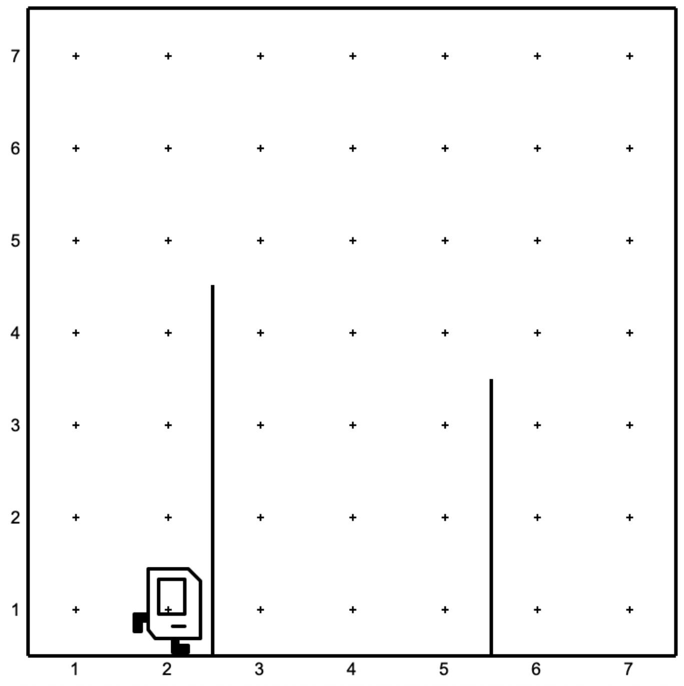
Write a program which will move Karel up the stem until the top (where the bottom left corner of the bloom should go), like so:



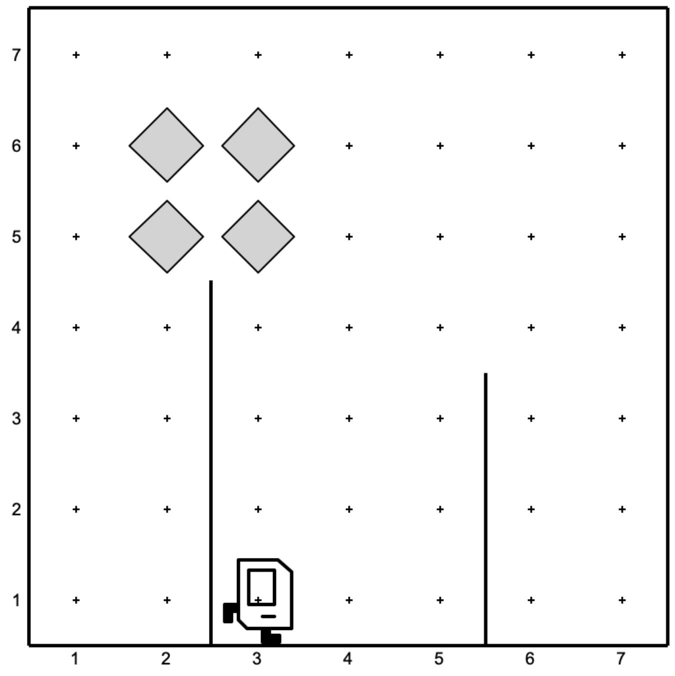
Note: if you run `put_beeper()` and it says “No beepers in bag”, your world file is likely out of date -- we've since fixed this bug! Save a copy of your code somewhere safe and then click the ... > Reset to Scaffold button in the top right to update everything automatically.

Spring flowers, Milestone 3: bloom flower (decomposition)

Assuming Karel starts facing the base of a stem, like so:



Write a program which will scale the stem, place 4 beepers in a bloom on top of the stem, and return Karel to face East at the bottom of the world directly right of the stem, like so:



If you've been following along with our milestones, you'll likely want to **use code from previous milestones** into helper functions to be called here! How can you make clever use of `bloom()`, `climb_stem()`, and `move_to_wall()` helper functions here?

Note: if you run `put_beeper()` and it says “No beepers in bag”, your world file is likely out of date -- we've since fixed this bug! Save a copy of your code somewhere safe and then click the ... > Reset to Scaffold button in the top right to update everything automatically.

Spring flowers, Milestone 4: bloom all the flowers!! (for loops, decomposition)

Write a program to make 2 spring flowers bloom! Karel will start facing East in the bottom left corner of the world, and Karel should end in the bottom right corner of the world facing East. Recall that there will always be 2 flower stems, but they are not guaranteed to be spaced the same width apart nor any predetermined height.

We've provided you two worlds on which to test your code. The `SpringFlowers1` world will load by default, but you can toggle to the `SpringFlowers2` world by changing the very last line in the file from `run_karel_program('SpringFlowers1.w')` to `run_karel_program('SpringFlowers2.w')` (and vice versa).

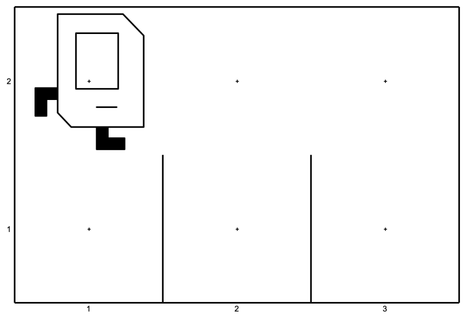
Note: if you run `put_beeper()` and it says “No beepers in bag”, your world file is likely out of date -- we've since fixed this bug! Save a copy of your code somewhere safe and then click the ... > Reset to Scaffold button in the top right to update everything automatically.

More problems

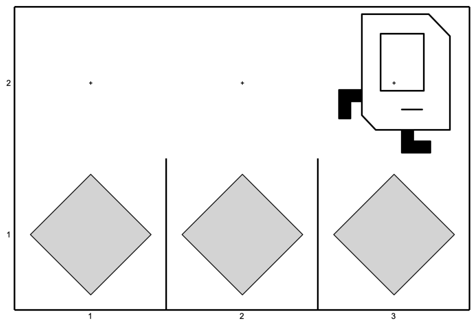
The below problems are for extra practice! We hope you have a good time with them.

Three slots (for loops, fencepost)

Karel is in the top left corner of a 2 row, 3 column world with three slots in the bottommost row, as such:



Write a program which will place beepers in each slot of the bottom row, so that the end world looks like this:



For an extra challenge, try writing this code with a `while` loop instead so that it works for worlds with any number of columns / slots!

Note: if you run `put_beeper()` and it says “No beepers in bag”, your `Slots.w` file is likely out of date -- we've since fixed this bug! Save a copy of your code somewhere safe and then click the ... > Reset to Scaffold button in the top right to update everything automatically.

10s across the board (for loops, while loops, fencepost)

Place 10 beepers in each position in the bottom row. Karel will begin in the bottom left corner of a world with no beepers; Karel should end in the bottom right corner of the world with 10 beepers across the bottom row (including the positions Karel starts and ends on!).

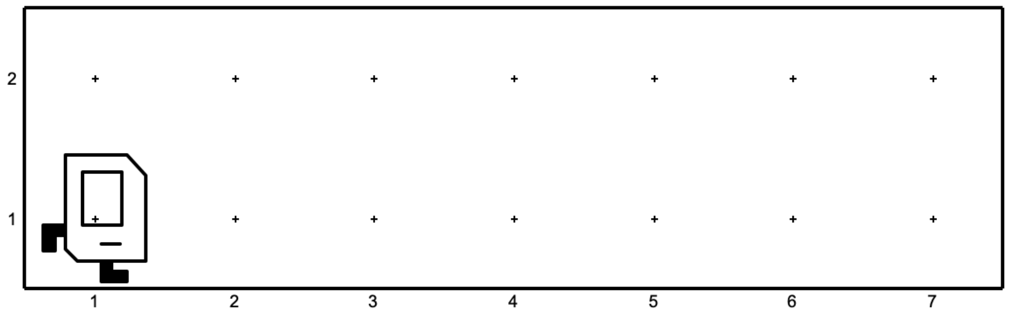
You could start by writing this assuming you know the number of columns (i.e. the length of each row) beforehand, and then tweak your code to make it work for any number of columns!

Test different sized worlds by changing the very last line in the file from `run_karel_program('TensKarel1.w')` to `run_karel_program('TensKarel2.w')` or `run_karel_program('TensKarel3.w')`

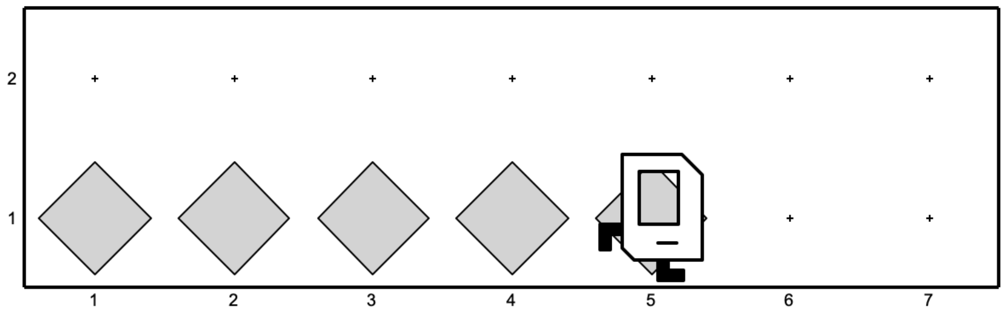
Maximum 5 (for loops, if/else)

Karel should place a maximum of 5 beepers across the first row of the world. If the world is less than 5 columns wide, Karel should fill the first row with beepers; if the world is more than 5 columns wide, Karel should only place 5 beepers.

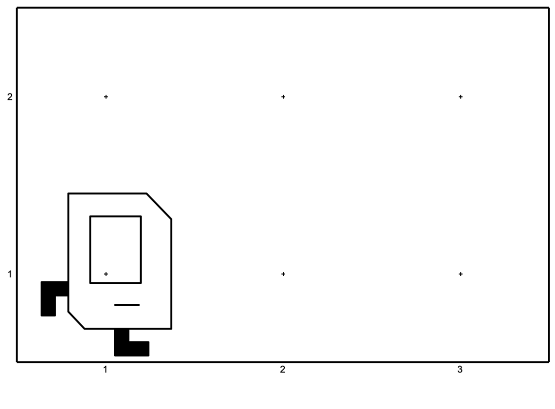
For example, for a world with 7 columns, like so:



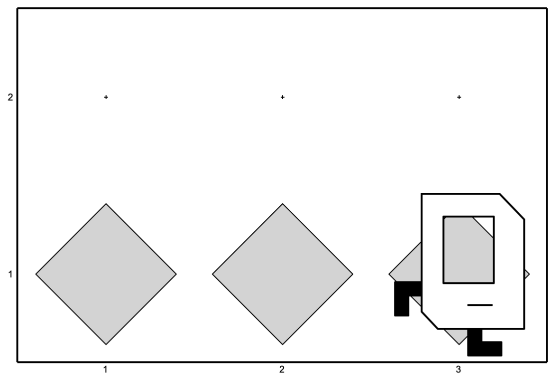
This should be the result:



And for a world with 3 columns, like so:



This should be the result:



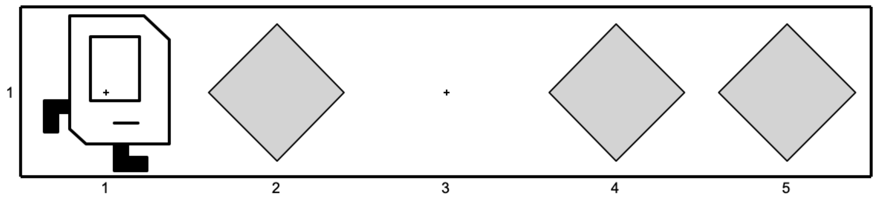
It may be simpler to approach this problem first assuming that the world will be greater than 5 columns wide, and then subsequently modifying your code to work for worlds that are only 3 columns wide.

Test different sized worlds by changing the very last line in the file from `run_karel_program('Max5Karel1.w')` to `run_karel_program('Max5Karel2.w')` or `run_karel_program('Max5Karel3.w')` or `run_karel_program('Max5Karel4.w')`

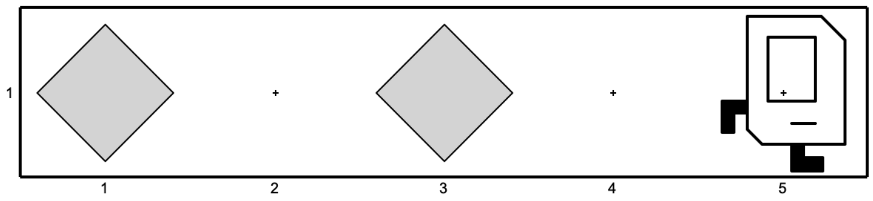
Invert (while loops, if/else)

Karel will be in a single row world with beepers in some positions. Karel should "invert" the pattern of the row -- pick beepers from the spots with beepers and place beepers from the empty spots -- and end facing East in the rightmost position. There will be no more than 1 beeper in each spot. Be sure to invert the positions Karel starts and ends on!

For example, if this is the initial world:



This should be the result:



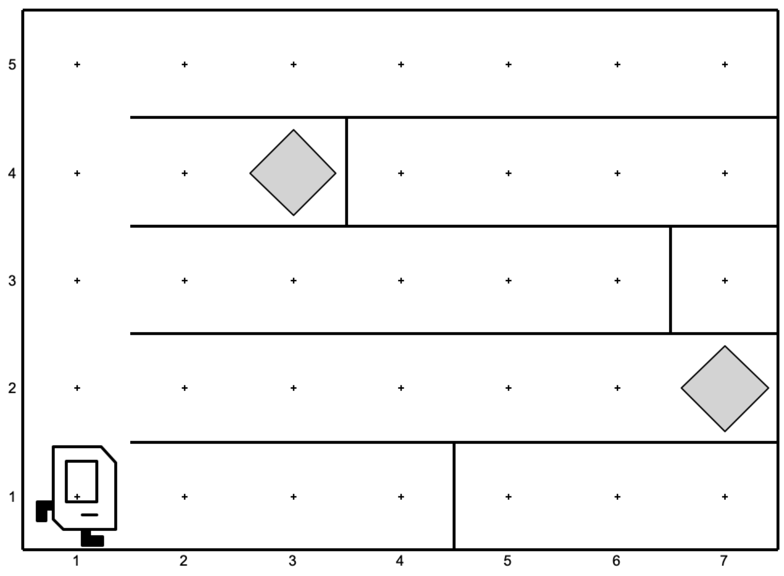
We've provided you two worlds on which to test your code. The `Invert1.w` world (pictured above) will load by default, but you can toggle to the `Invert2.w` world by changing the very last line in the file from `run_karel_program('Invert1.w')` to `run_karel_program('Invert2.w')` (and vice versa).

Note: if you run `put_beeper()` and it says “No beepers in bag”, your world file is likely out of date -- we've since fixed this bug! Save a copy of your code somewhere safe and then click the ... > Reset to Scaffold button in the top right to update everything automatically.

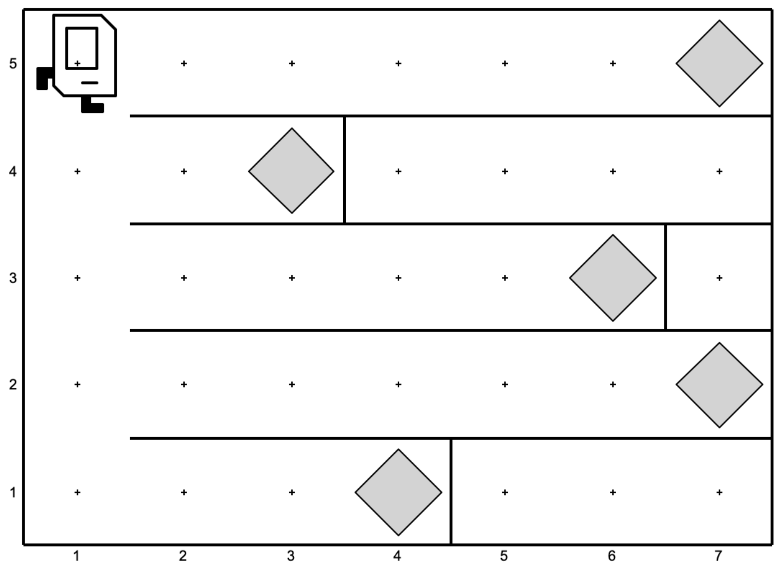
If you'd like to make your own world for Karel, change the last line of the file to `run_karel_program()` and use the Karel editor pane to change the world dimensions. To teleport Karel to a different spot, place and remove beepers, or change Karel's direction, right click on the world and select what you'd like to do, and then click a spot in the world to execute that action.

Five corridors (for loops, while loops, fencepost, if/else)

Karel will begin in the bottom right corner of a world with 5 "corridors", some of which have beepers at the end of them and some of which are empty, like so:



Get Karel to traverse each corridor (essentially, each row, until a wall blocks her path). At the end of each corridor (they may be different lengths!), if a beeper isn't already there, put a beeper down. Karel should end up at the beginning of the topmost corridor (highest row number), and each corridor should have a beeper at the end of it, like so:

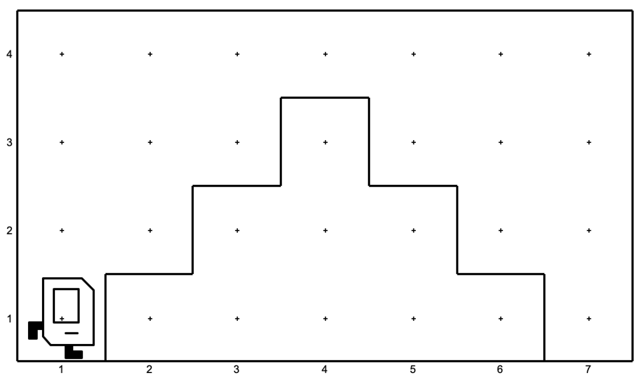


Note: if you run `put_beeper()` and it says “No beepers in bag”, your world file is likely out of date -- we've since fixed this bug! Save a copy of your code somewhere safe and then click the ... > Reset to Scaffold button in the top right to update everything automatically.

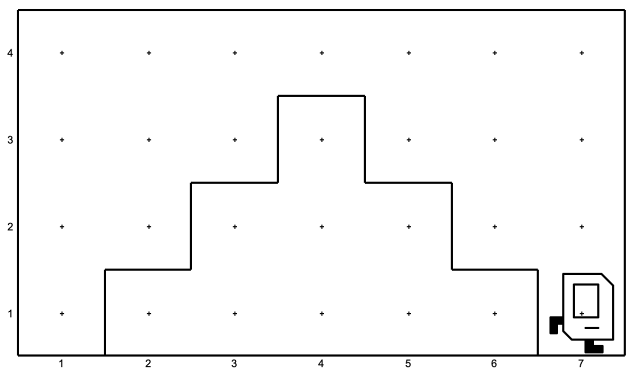
If you'd like to make your own five-corridor world for Karel, you can use the Karel editor pane to change the world dimensions. To teleport Karel to a different spot, place and remove beepers, or change Karel's direction, right click on the world and select what you'd like to do, and then click a spot in the world to execute that action.

Upstairs, downstairs (for/while loops)

Karel will be in the bottom left corner of a world with a staircase of 3 steps up and then 3 steps down, like this:



Write a program to get Karel to climb up the stairs and then down, so that Karel ends up in the bottom right corner of the world like this:

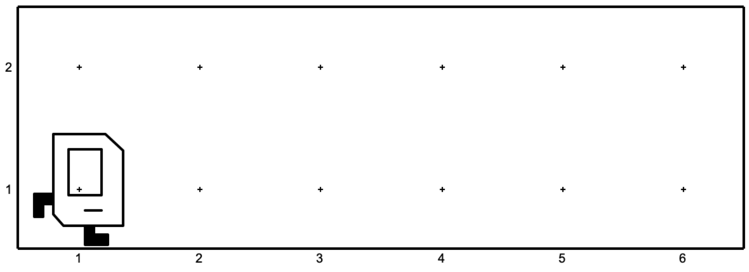


You may find it helpful to write a function which moves Karel up a single step.

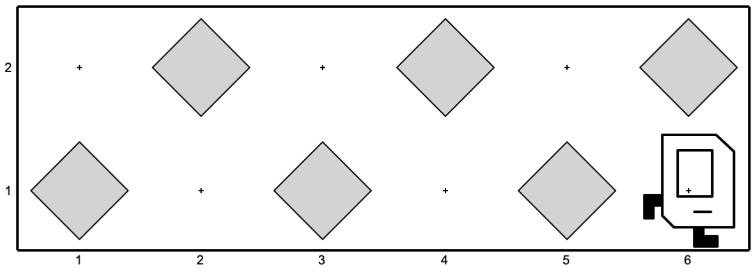
Zig zag (while loops, optional if/else)

Karel will be in a blank world at the bottom left corner. Karel should place a zig zag pattern of beepers such that all odd columns have beepers in row 1 and all even columns have beepers in row 2.

For example, if Karel starts in a blank world with 2 rows and 6 columns, like so:

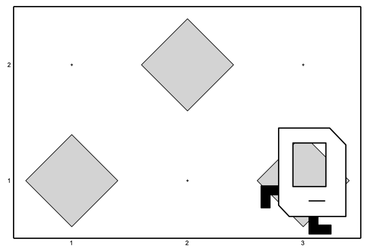


The end result should be (note that it doesn't really matter where Karel ends up, though as a hint this is where our solution places her):



Start by assuming that all worlds will be 2 rows tall and an even number of columns.

After you've done that, if you'd like an extra challenge and some more practice with conditional logic, try writing your code so that it works (i.e. Karel doesn't attempt to walk through any walls, and the pattern is placed correctly) for all worlds that are 2 rows tall and any number of columns (including odd numbers of columns, including a single column!). For example, if Karel starts in a blank world with 2 rows and 3 columns, the end result should be (again, it doesn't matter where Karel ends up, so long as the beepers are in the right place and Karel doesn't try to walk through a wall, though as a hint this is where our solution places her):



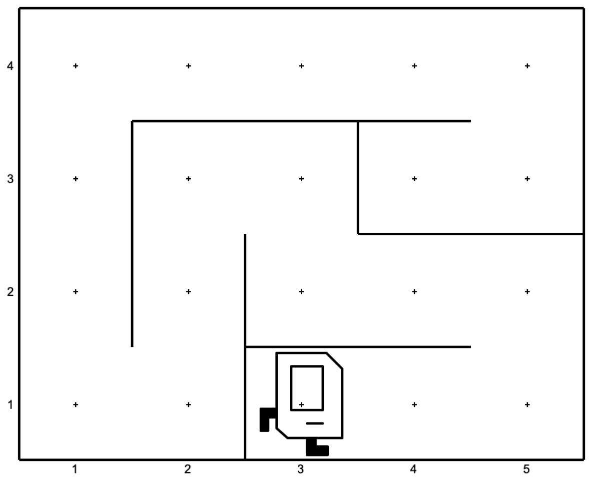
We've provided you both of the worlds pictured above on which to test your code. The 6x2 world will load by default, but you can toggle to the 3x2 world by changing the very last line in the file from `run_karel_program('ZigZag1.w')` to `run_karel_program('ZigZag2.w')` (and vice versa).

If no beepers are present, make sure the beeper bag in the world files is set to exactly `BeeperBag: INFINITY`

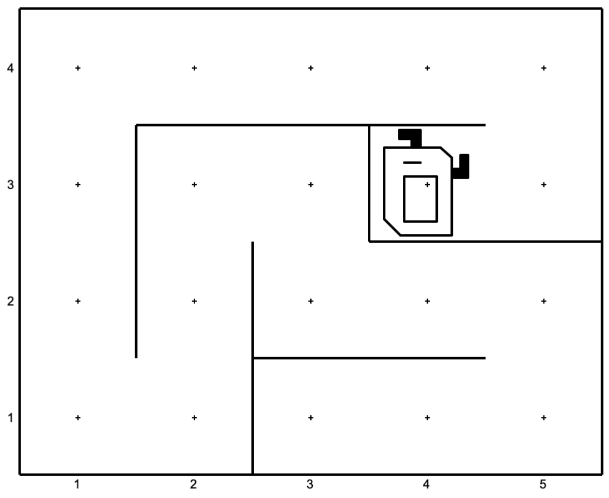
Labyrinth (while loops, if/else)

According to Wikipedia, "*maze* refers to a complex branching multicursal puzzle with choices of path and direction, while a unicursal *labyrinth* has only a single path to the center. A labyrinth in this sense has an unambiguous route to the center and back and presents no navigational challenge."

We've designed a labyrinth Karel world -- Karel will start in one dead end, and so long as Karel moves forwards through the labyrinth, Karel will eventually end up at the other dead end. For example, if Karel starts like this:



Karel should end like this (though it doesn't really matter what way Karel faces):



Write a program which will allow Karel to solve any labyrinth. Karel will always start facing the open direction.

We've provided you two worlds on which to test your code. The `Labyrinth1.w` world (pictured above) will load by default, but you can toggle to the `Labyrinth2.w` world by changing the very last line in the file from `run_karel_program('Labyrinth1.w')` to `run_karel_program('Labyrinth2.w')` (and vice versa).

We recommend you pause here and think about how you might approach this problem before reading on. If you don't want any hints about how to solve this, stop reading entirely, and have fun!

How might you approach this problem? If you put yourself into Karel's shoes, you'll quickly find that solving a labyrinth means moving until you hit a wall, reorienting yourself to turn whichever way is open (but not going back the way you came!), moving until you hit a wall, and repeating until you end up in a dead end.

We recommend you approach this problem by writing a `move_to_wall()` helper function and a `find_direction()` helper function which will check left and right in order to get Karel to face whichever direction which isn't blocked off by a wall. If both directions are blocked, it doesn't matter which way Karel faces.

Treasure hunt, part 1 (while loops)

Treasure hunter extraordinaire Karel has been given a map to a treasure. The map has specific logical rules -- Karel should move until she encounters a beeper. When Karel encounters a beeper, she should turn left, and move until she encounters a beeper, etc. until she is standing on the final treasure: a pile of 10 beepers. Because the treasure is meant for Karel only, Karel should pick up all the beepers she encounters as she moves along, including the final pile of 10. Karel should end up where the pile of 10 beepers is, but there should be no beepers left in the world.

We recommend you write a `move_to_beeper()` helper function which will move Karel forwards until she's standing on a beeper to help you with this problem.

Treasure hunt, part 2 (while loops, if/else)

If you've solved the first treasure hunt and you're in the mood to write more code and find more treasure -- you're in luck: Karel has a second treasure map, for another treasure of 10 beepers! The rules for this map are slightly more complicated than the previous one.

Similarly to the previous map, Karel should move until she encounters a beeper. When Karel encounters a beeper, she should *turn left if her left side is clear (no walls present)*, but *if there is a wall in the way she should turn right,* and then move forward until another beeper, etc. until the final pile of 10 beepers!

Once again, we recommend you write a `move_to_beeper()` helper function which will move Karel forwards until she's standing on a beeper to help you with this problem.