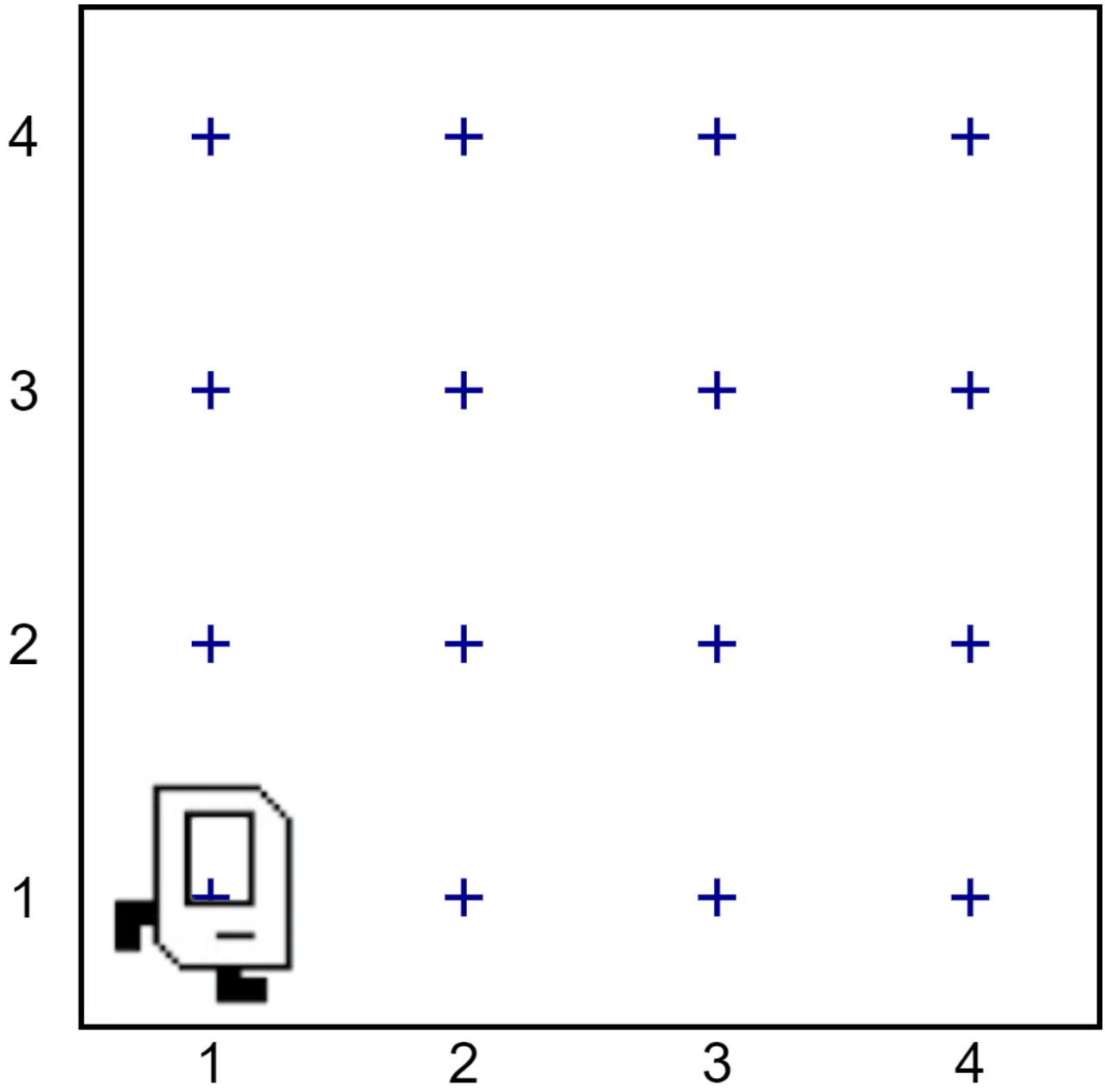
Chapter 5: For Loops

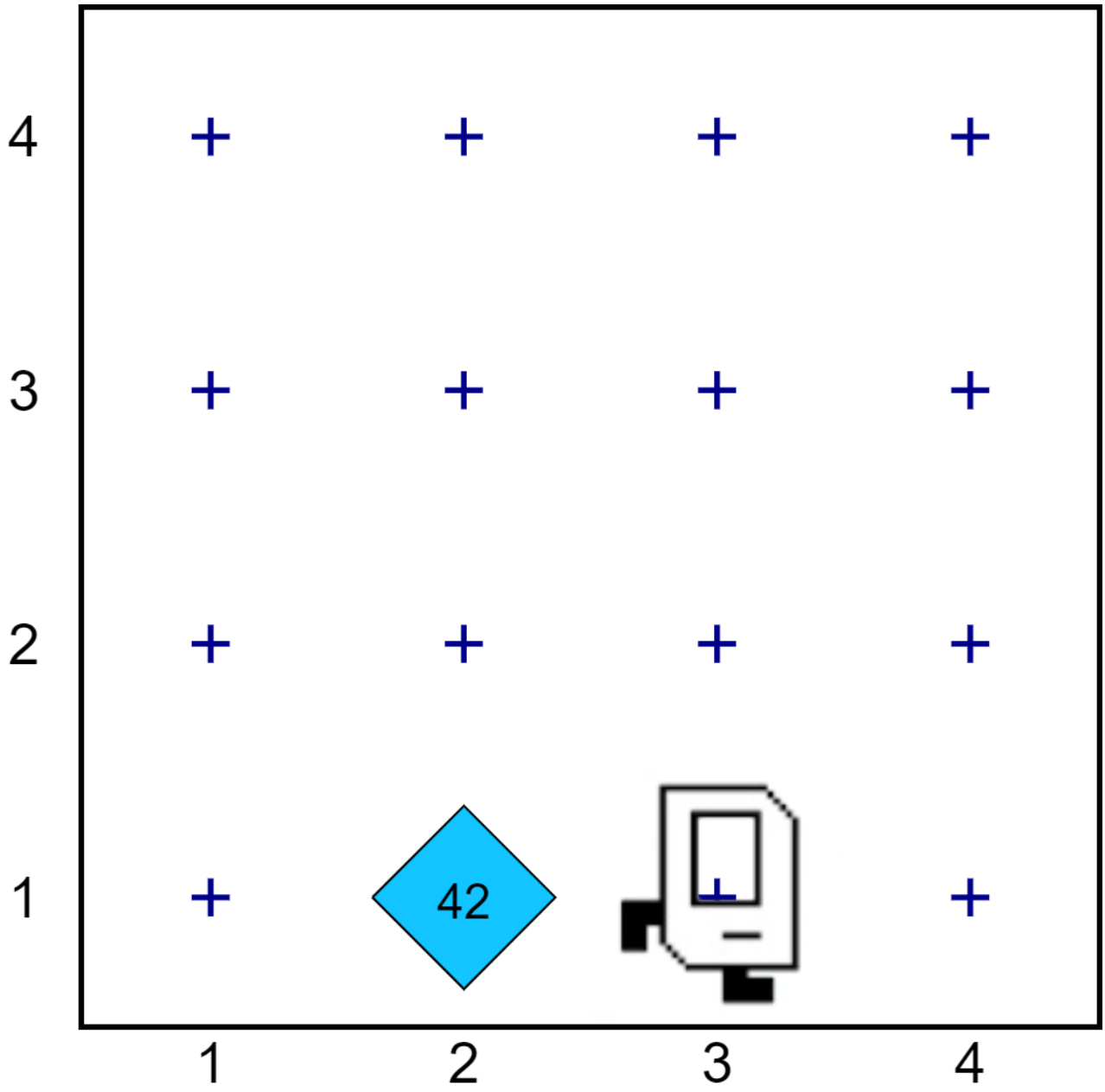
compedu.stanford.edu/karel-reader/docs/python/en/chapter5.html

One of the things that computers are especially good at is repetition. How can we convince Karel to execute a block of code multiple times? To see how repetition can be used, consider the task of placing 42 beepers:

Before:



After:



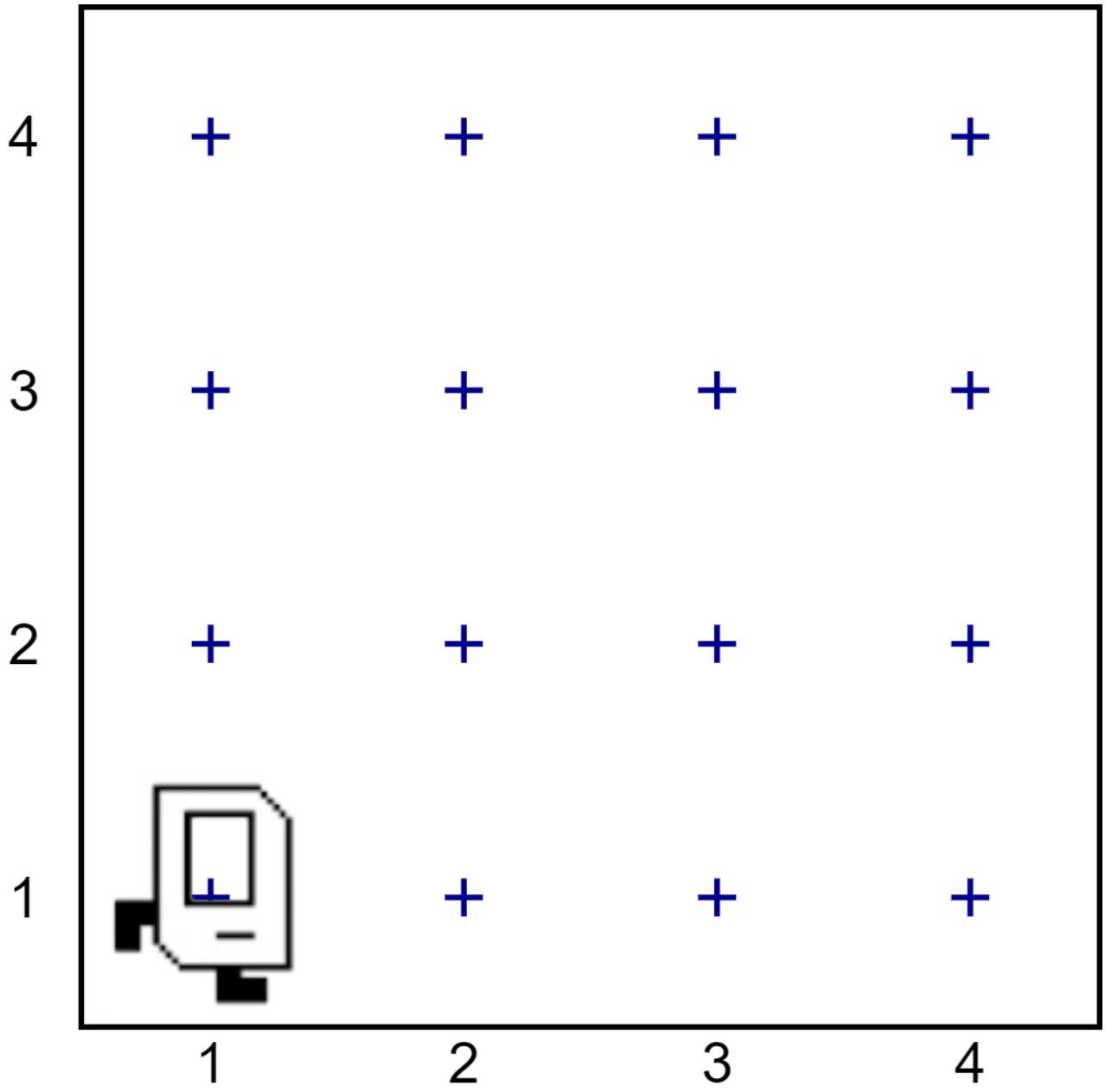
Basic For Loop

Since you know that there are exactly 42 beepers to place, the control statement that you need is a **for loop**, which specifies that you want to repeat some operation a fixed number of times. The structure of the `for` statement appears complicated primarily because it is actually much more powerful than anything Karel needs. The only version of the `for` syntax that Karel uses is:

`for i in range(` *count* `):`
statements to be repeated

We will go over all the details of the `for` loop later in the class. For now you should read this line as a way to express, "repeat the statements in the function body *count* times." We can use this new **for loop** to place 42 beepers by replacing *count* with 42 and putting the command `put_beeper()` inside of the `for` loop code block. We call commands in the code block the **body**:

```
# File: PlaceManyBeepers.py
# ---------------------------
# Places 42 beepers using a for loop
from karel.stanfordkarel import *
def main():
move()
# repeat put_beeper many times
for i in range(42):
put_beeper()
move()
```



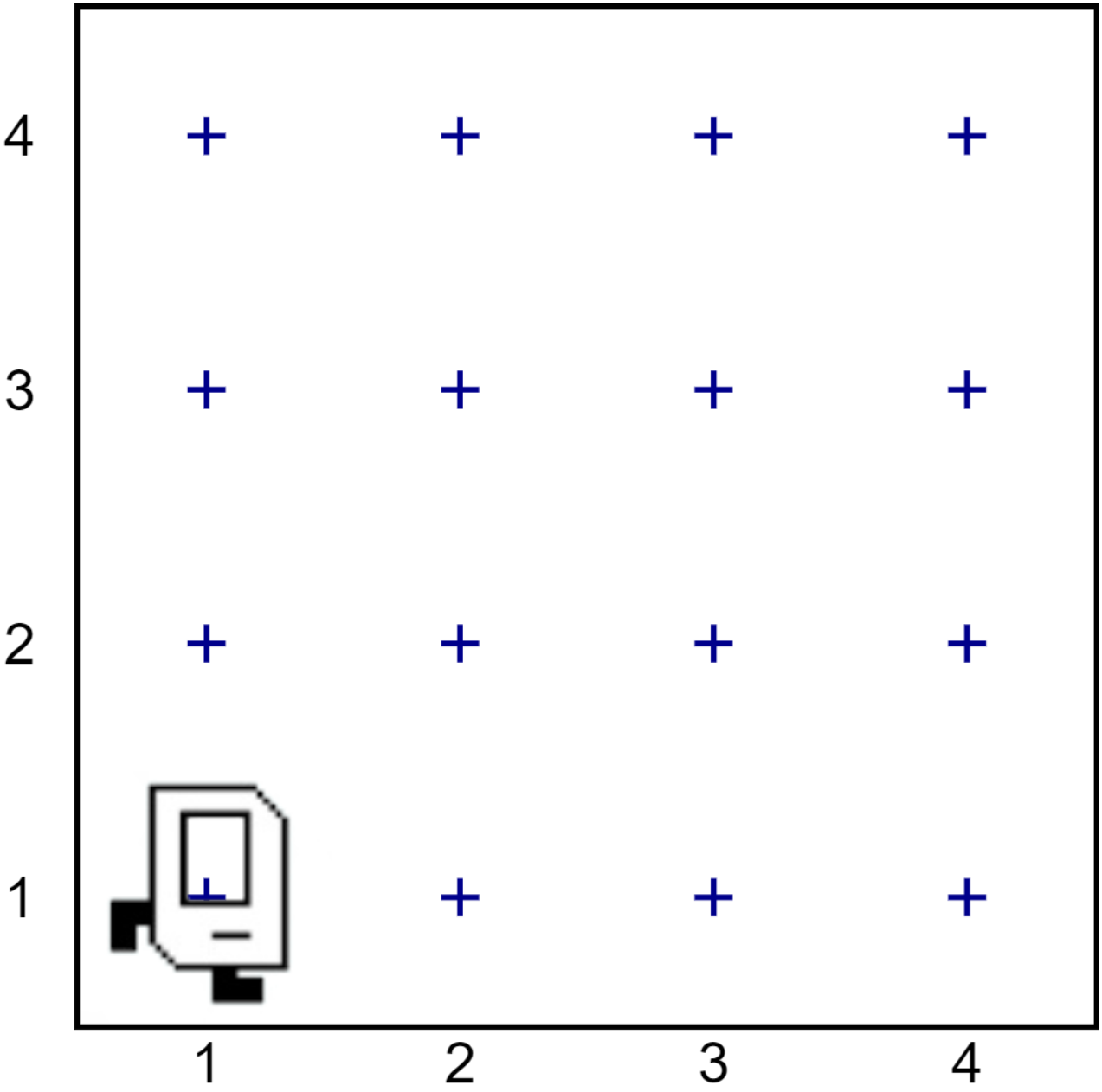
The code above is editable. Try to change it so that it places only 15 beepers.

Matching Postconditions with Preconditions

The previous example gives the impression that a `for` loop repeats a single line of code. However the body of the `for` loop (the statements that get repeated) can be multiple lines. Here is an example of a program that puts a beeper in each corner of a world:

```
# File: CornerBeepers.py

# ---------------------------

# Places one beeper in each corner

from karel.stanfordkarel import *

def main():
```

```
# repeat the body 4 times
for i in range(4):
    put_beeper()
    move()
    move()
    move()
    turn_left()
```



Pay very close attention to the way that the program flows through these control statements. The program runs through the set of commands in the `for` loop body one at a time. It repeats the body four times.

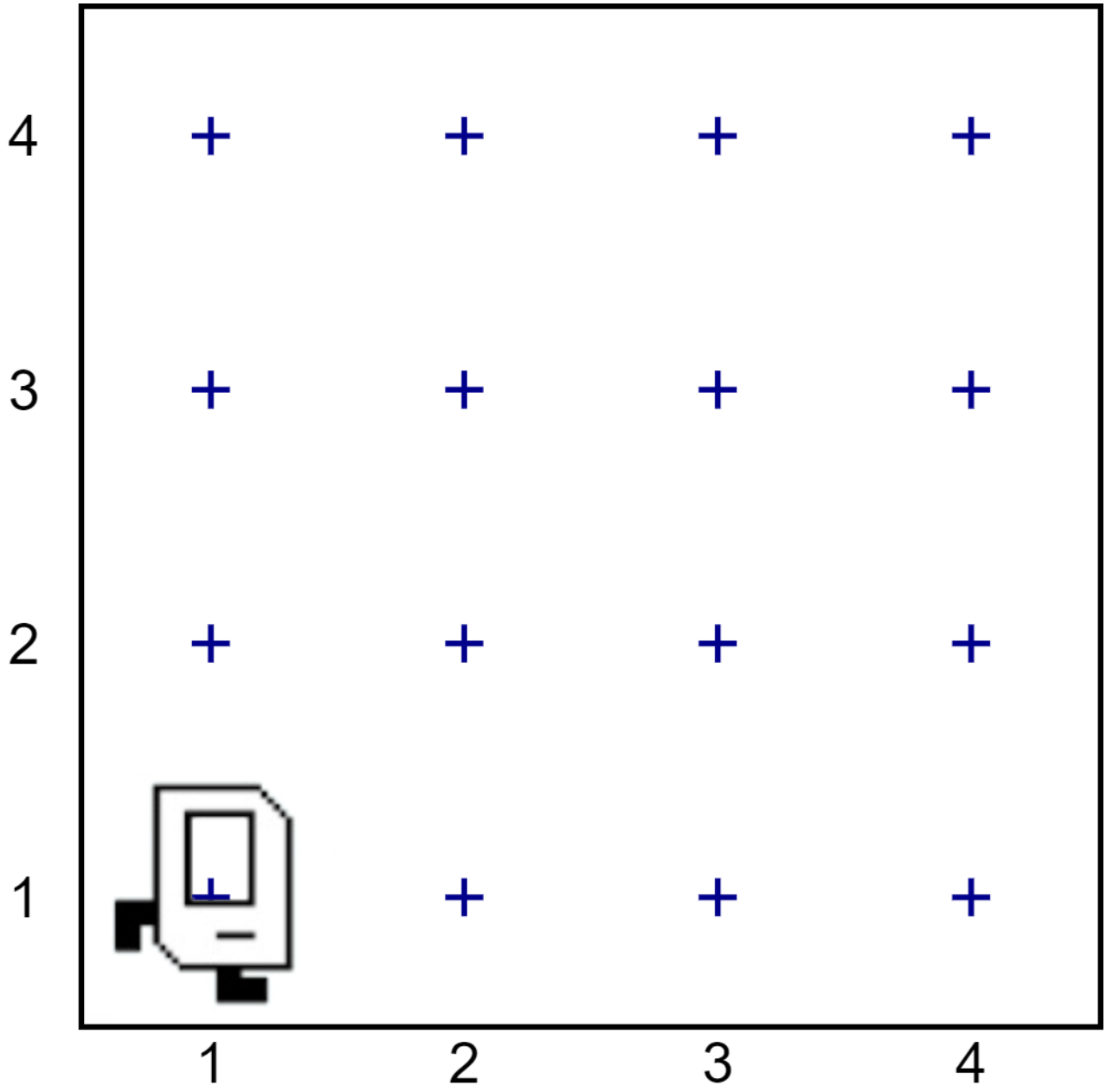
Perhaps the single most complicated part of writing a loop is that you need the state of the world at the end of the loop (the **postcondition**) to be a valid state of the world for the start of the loop (the **precondition**). In the above example the assumptions match. Good times. At the start of the loop, Karel is always on a square with no beepers facing the next empty corner. What if you deleted the `turn_left()` at the end of the loop? The postcondition at the end of the first iteration would no longer satisfy the assumptions made about Karel facing the next empty corner. The code is editable. Try deleting the `turn_left()` command to see what happens!

Nested Loops

Technically the body of a `for` loop can contain any control flow code, even other loops. Here is an example of a `for` loop that repeats a call to a function which also has a `for` loop. We call this a "nested" loop. Try to read through the program, and understand what it does, before running it:

```
# File: CornerFiveBeepers.py

# ---------------------------

# Places five beepers in each corner

from karel.stanfordkarel import *

def main():

# Repeat once for each corner

for i in range(4):

put_five_beepers()

move_to_next_corner()

# reposition karel to the next corner

def move_to_next_corner() :

move()

move()

move()

turn_left()

# places 5 beepers using a for loop

def put_five_beepers() :

for i in range(5):
```

put_beeper()



Next Chapter