CS10 NEWS

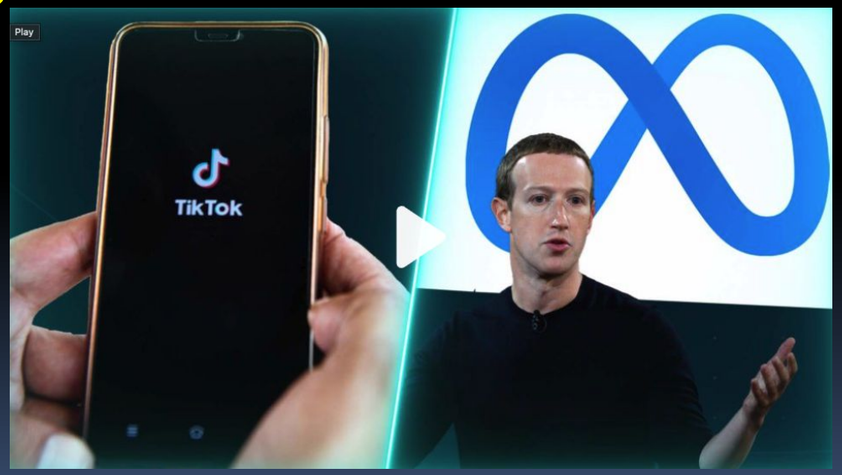
The Beauty and Joy of Computing

Testing, Project 3
Mutable vs Immutable Data

UC Berkeley
Teaching Professor
Dan Garcia

Why deleting something from the internet is “almost impossible”

Most people may live out their digital lives with the assumption they can delete their posts, messages and personal data from services whenever they choose. But a tech hearing this week threw that core assumption into question. ... “Just assume that everything you put out there can be used by anyone, and will live in perpetuity,”

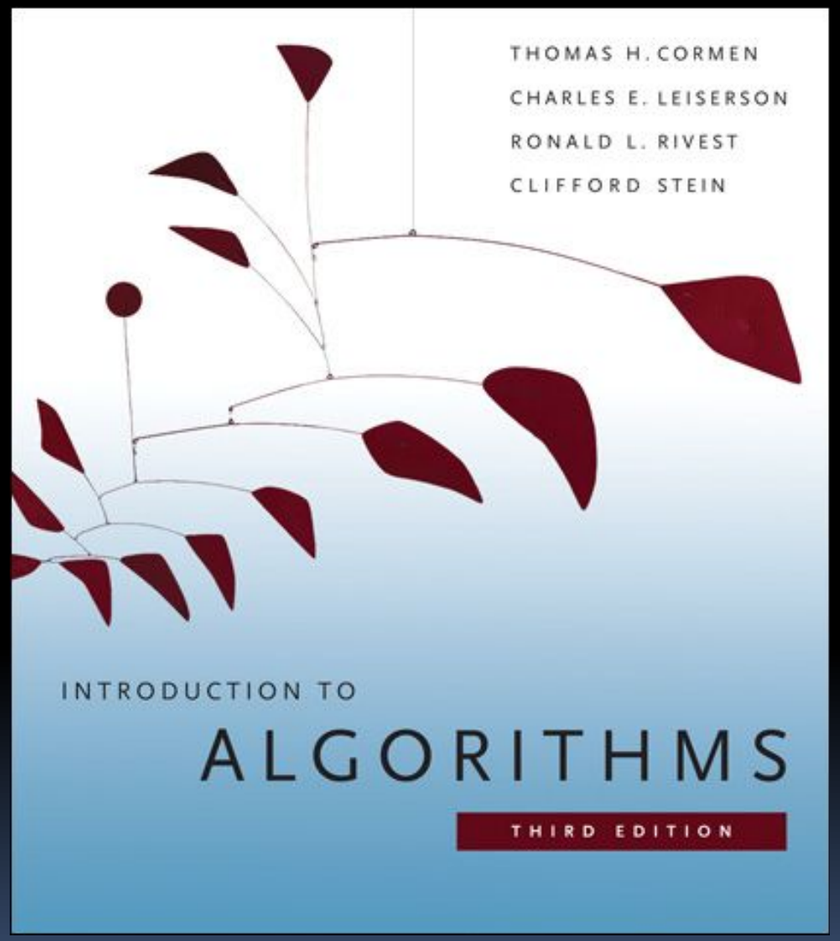


www.cnn.com/2022/09/18/tech/deleting-data/

Algorithms: Correctness, Summary

Reference Text for Algorithms

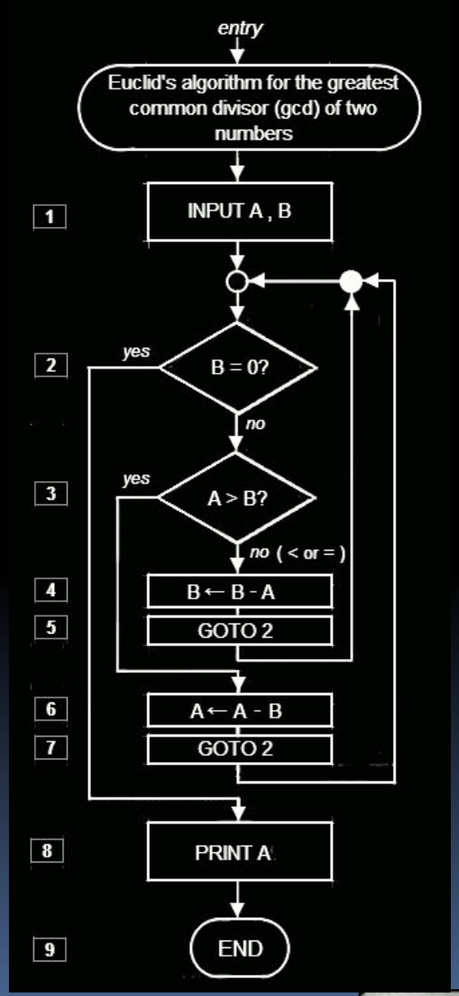
- This book launched a generation of CS students into Algorithm Analysis
  - It's on everyone's shelf
  - It might be hard to grok at this point, but if you go on in CS, remember it & own it!
  - …but get the most recent version



Algorithm Analysis: Is Algorithm Correct?

- An algorithm is correct if, for every input, it reports the correct output and doesn't run forever or cause an error.
  - Incorrect algorithms may run forever, or may crash, or may not return the correct answer.
    - They could still be useful!
    - Consider an approximation…
  - For now, we'll only consider correct algorithms

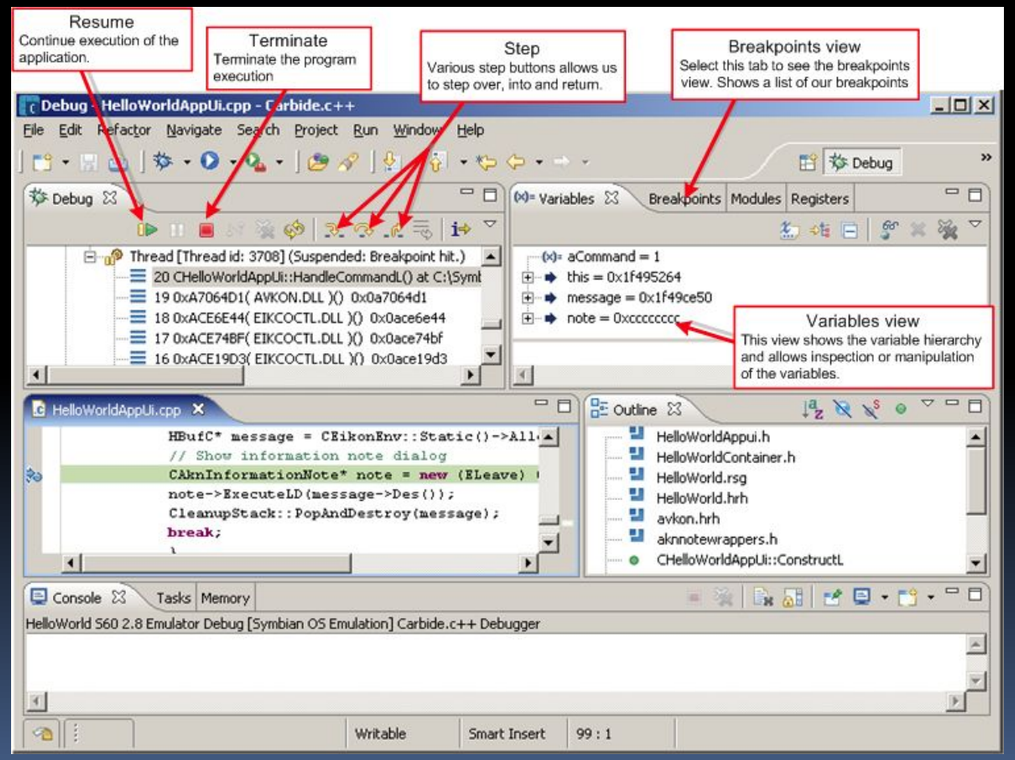
Euclid's GCD Algorithm
(*Wikimedia*)



Garcia

How do you know if it is correct?

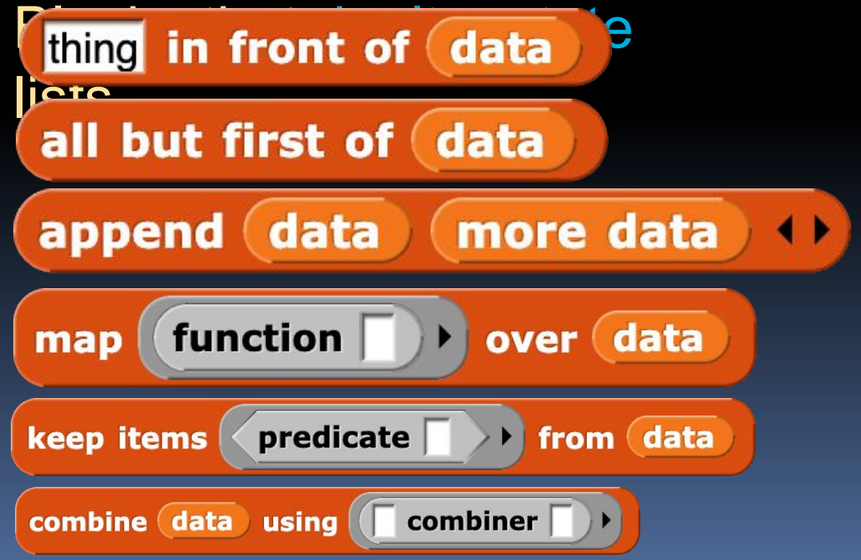
- For algorithms?
  - Reasoning formally or mathematically
- For programs? Empirically via testing:
  - Unit Testing
  - Debugging
  - Can <u>never</u> be sure algorithm is correct by testing implemention



Testing, HW3 Mutable vs Immutable Data

Mutable vs Immutable Data: Intro

- Immutable data
  - Data whose ”state” can’t be changed once created
  - This is important with concurrency, when there are many different workers using the data at the same time
  - Blocks that create lists



- Mutable data
  - Data whose “state” can be changed once created
  - Harder to get concurrent programs working correctly!
  - Blocks that mutate lists

Mutable vs Immutable Data: Example

- Immutable data

```
script variables numbers doubled numbers
set numbers to numbers from 1 to 3
set doubled numbers to Immutable Double List numbers
say join original: numbers doubled: doubled numbers
```

```
Immutable Double List data :
report map Double over data
```

```
Double n #
report 2 × n
```

original:123
doubled:246

- Mutable data

```
script variables numbers
set numbers to numbers from 1 to 3
Mutable Double List numbers
say join numbers
```

```
Mutable Double List data :
for i = 1 to length of data
    replace item i of data with Double item i of data
```

246

Monolithic code … is it good or bad?

Monolithic code

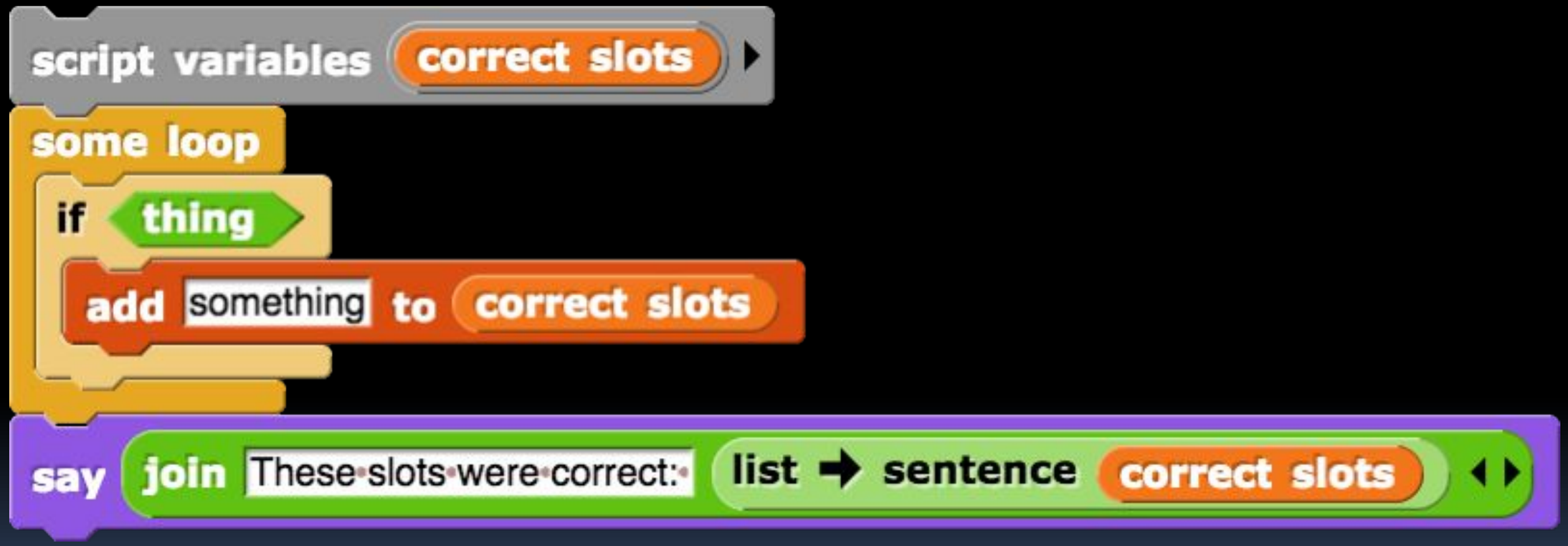
Monolithic code is:

- Hard to write
- Hard to debug
- Hard to read

Getting help from classmates, lab assistants, and TAs is an important part of coding practice. Therefore, making it quick and easy for them to help you is crucial!

Good abstractions

Let's say you have something like this in your HW1:



Is there another way that could help with debugging?

Project 1 abstraction

Consider the following version:

This way, all code related to finding the matching slots is in here… and not in the top level, where it contributes to monolithic code.

Checking correctness of this functionality in the program is much easier now!

When to test?

A. At the beginning, before you write code
B. In the middle, after you've written a bit
C. At the end once you think you're done
D. All of the above
E. None of the above

When poll is active, respond at **pollev.com/ddg**

Text **DDG** to **22333** once to join

L09 When to test?

At the beginning, before you write code

In the middle, after you've written a bit

At the end, once you think you're done

All of the above

None of the above

What is testing?

Trying to make sure that your program is correct. Some approaches:

- Check each block individually for correctness according to its *spec* with example inputs and outputs: "unit test"
- Do unexpected things to your blocks and program (answer an "ask" block incorrectly, or click the wrong buttons)
- Run your entire program to see if everything is working and makes sense: "integration test"
- Test as if you know what's inside your code "glass box testing"
- Test as if you have no idea what's inside "black box testing"
- Add feature and go through all the old tests "regression testing"

Test-driven development

Write tests first! Now, you can use them to help you quickly find out if what you have written is correct.

Test-driven development

An example of writing simple, yet effective tests

We want to write the block 

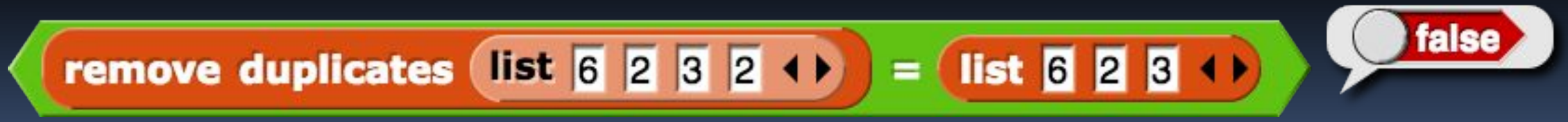
, which takes a list as input and reports a **new list** with any duplicate values removed. That is, the **first** instance of a duplicate value is kept and all others will not be present in the output.

First write tests…

Test-driven development

A **unit test**

When the block isn't correct…



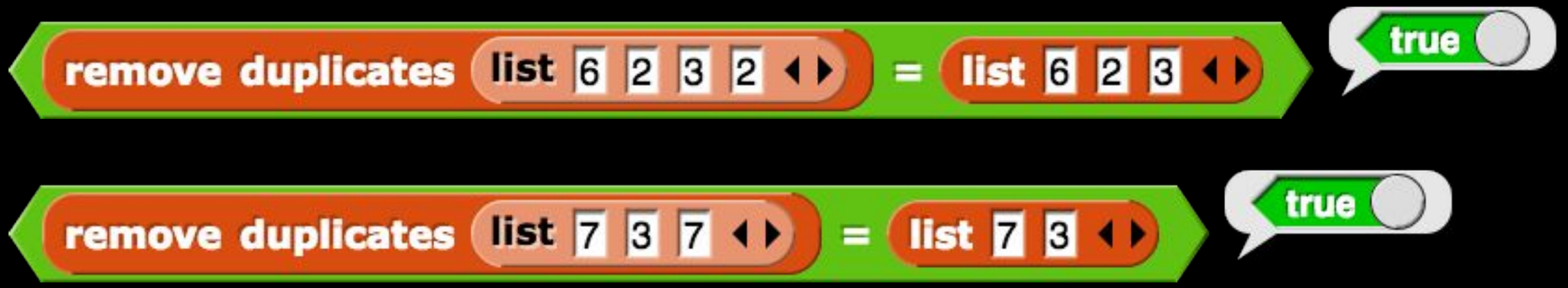
Unit tests

Is the block correct now?

Unit tests

Is the block correct now?

Unit tests



Is the block correct now?

Unit tests

remove duplicates list 6 2 3 2 = list 6 2 3 — true

remove duplicates list 7 3 7 = list 7 3 — true

Is the block correct now?

+remove+duplicates+ data ⋮ +

report all but last of data

Hmm… How can we improve the tests?

Unit tests

Add more cases!

remove duplicates list 6 2 3 2 = list 6 2 3

remove duplicates list 2 2 8 = list 2 8

remove duplicates list 4 4 4 4 = list 4

More test cases? Brainstorm to make sure you cover as many of the edge cases as you can think of…

Testing in Project 1

matching "green" slots between guess: [ ] and secret: joy = ---

matching "green" slots between guess: cal and secret: joy = ---

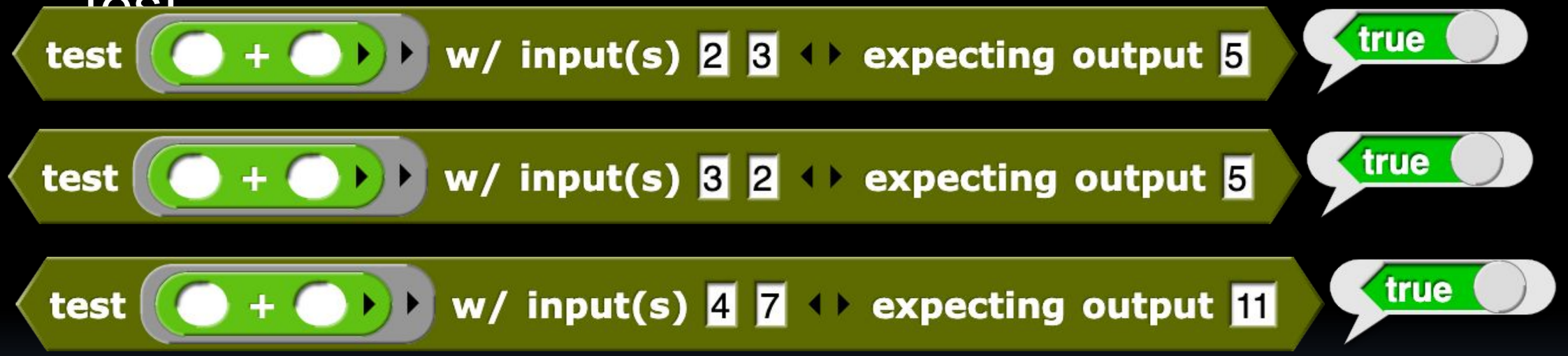
matching "green" slots between guess: jay and secret: joy = j-y

matching "green" slots between guess: to and secret: joy = -o-

matching "green" slots between guess: soylent and secret: joy = -o-

testing block

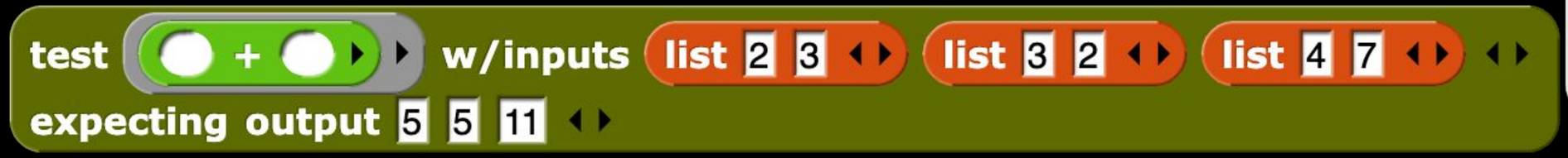
This block lets you test a function once with inputs and the expected output, returning true when it passes the test

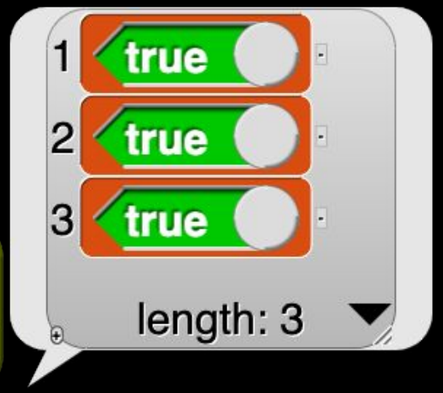


Note that you click the right arrow to put in the second input to plus … the number of inputs has to match the blank "holes" in the function definition.

testing blocks (1/2)

…but don't want to click each test manually! Introducing…





Now you put each set of inputs in a separate list

Note that if you have a function that takes a *single* input, you have to wrap each input in a list block.

testing blocks (2/2)

Now instead of having to click these…

remove duplicates list 6 2 3 2 = list 6 2 3

remove duplicates list 2 2 8 = list 2 8

remove duplicates list 4 4 4 4 = list 4

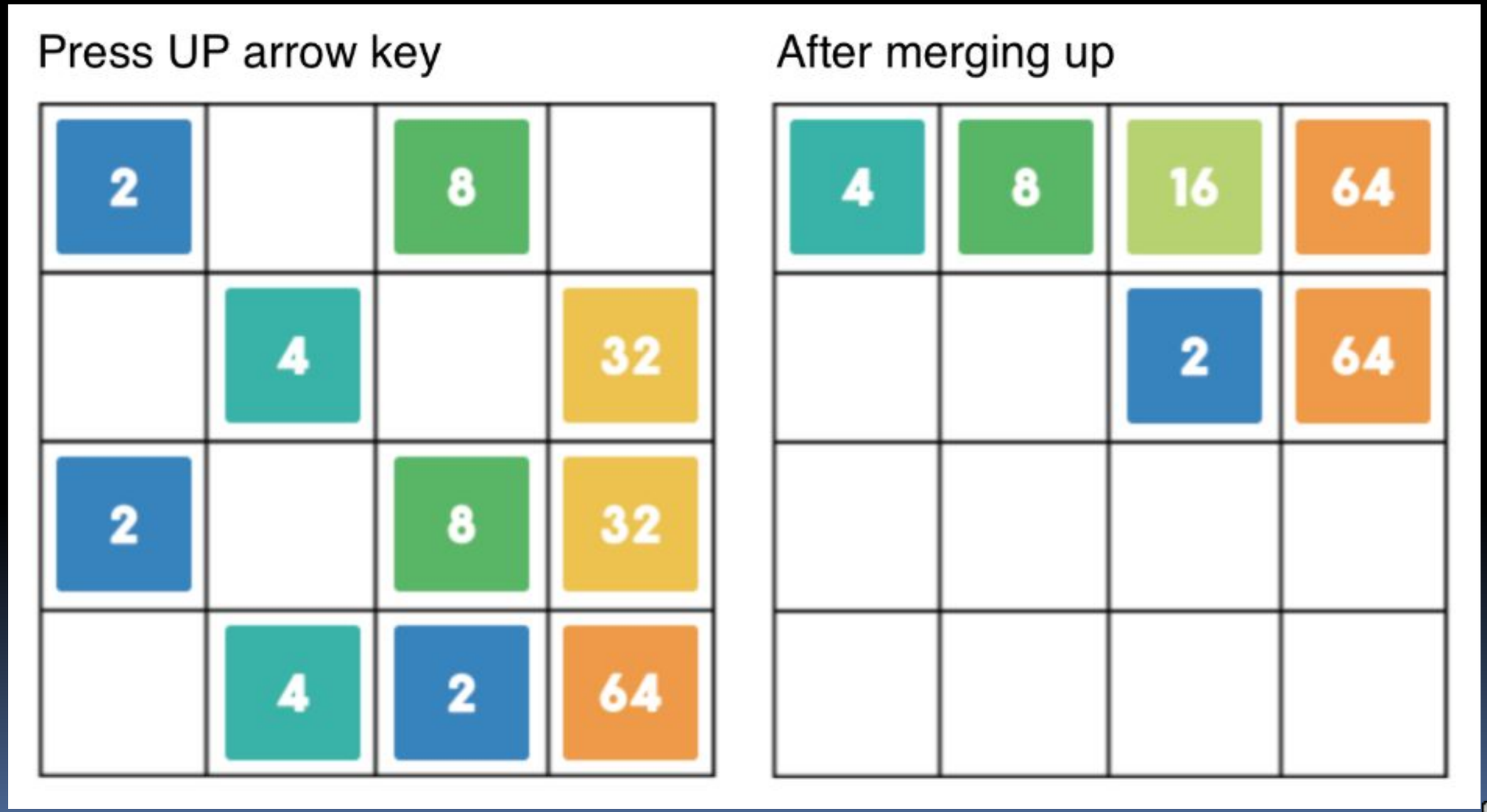
We have one block that covers ALL our test cases!

We obviously want all true values…

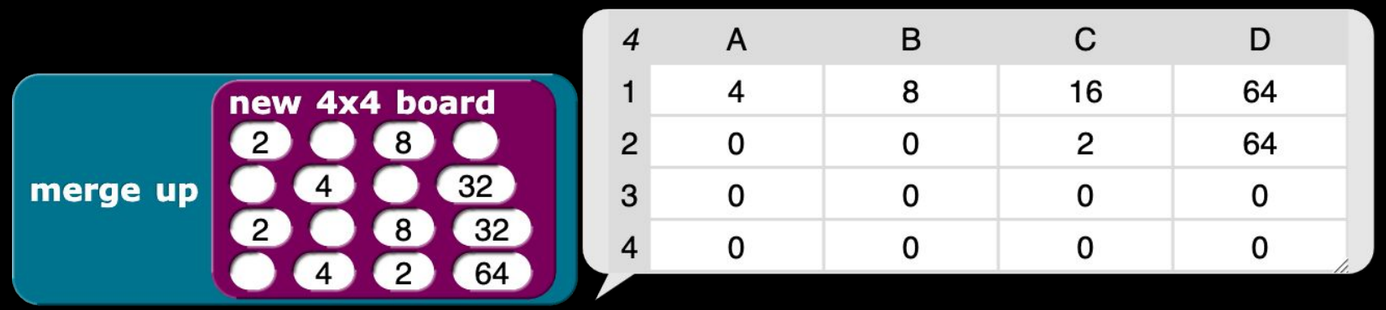
test remove duplicates w/inputs
list list 6 2 3 2 list list 2 2 8 list list 4 4 4 4
expecting output list 6 2 3 list 2 8 list 4

1 false
2 true
3 false
length: 3

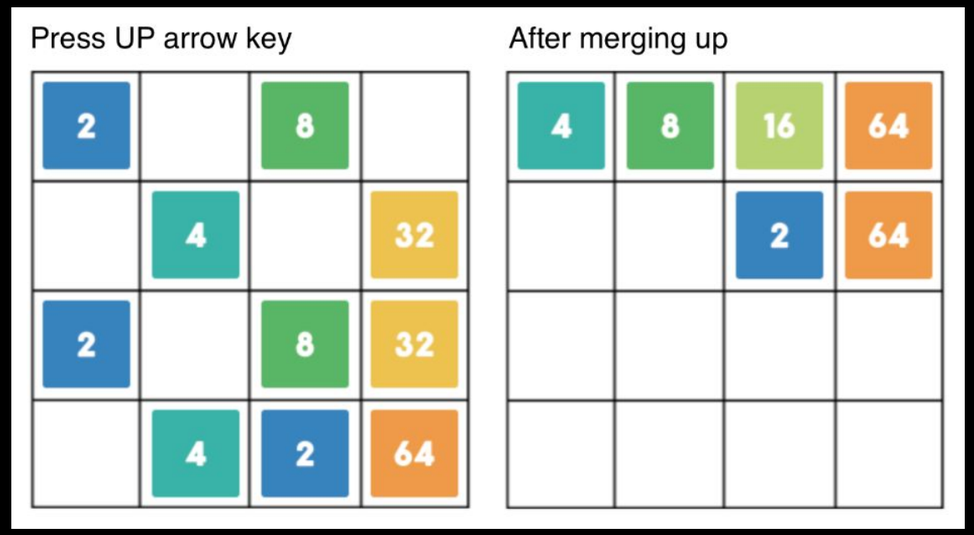
2048 introduction: `play2048.co`

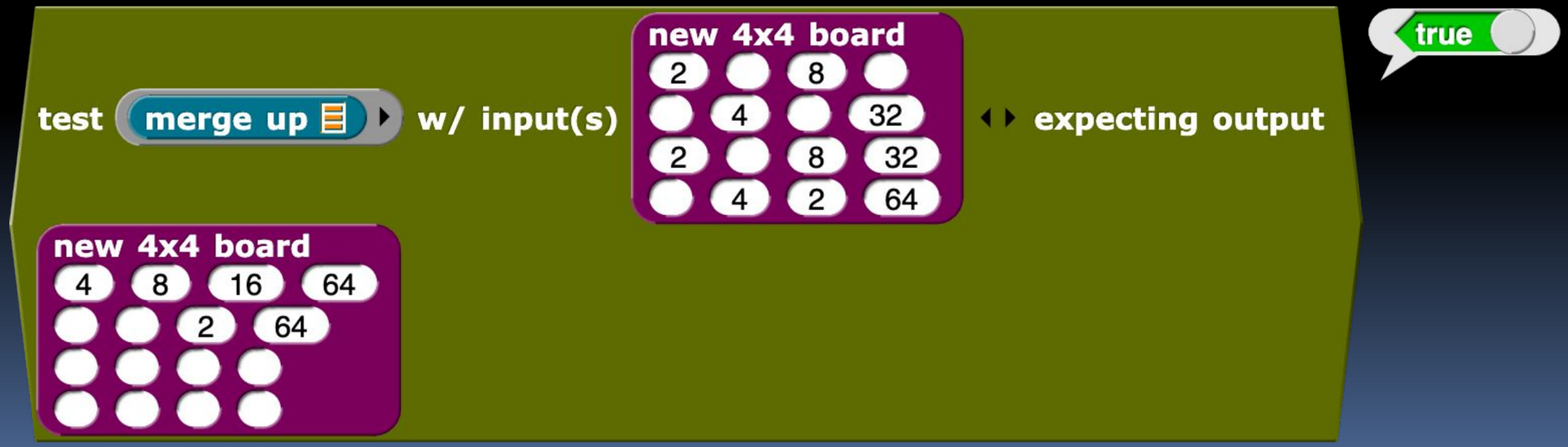


2048 unit test

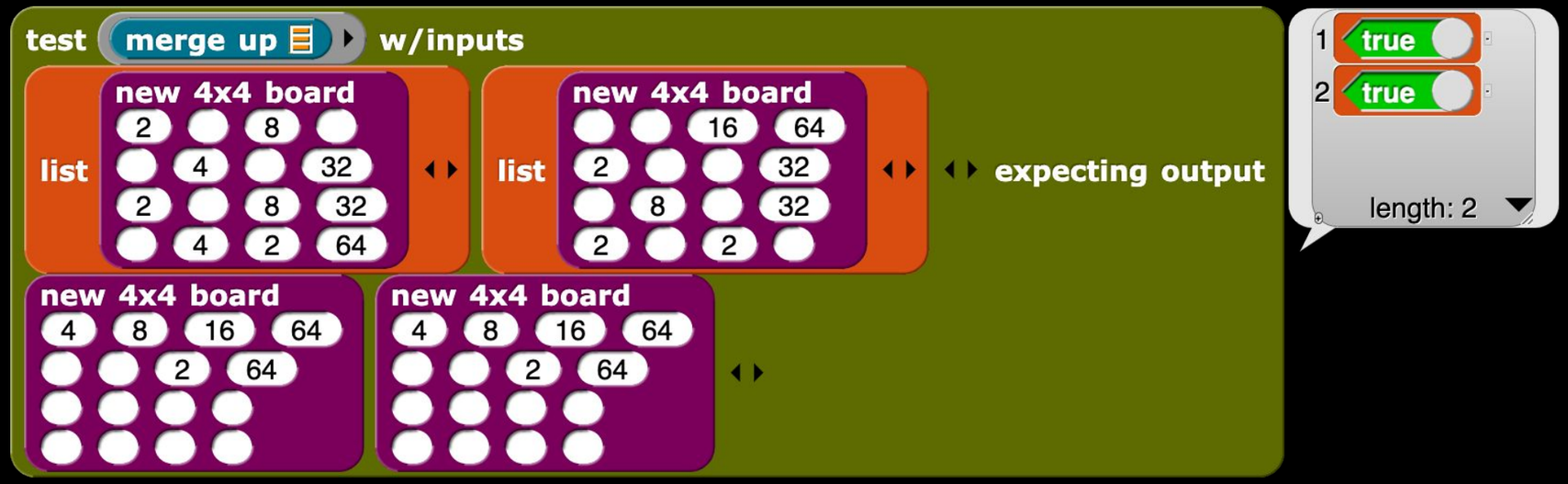


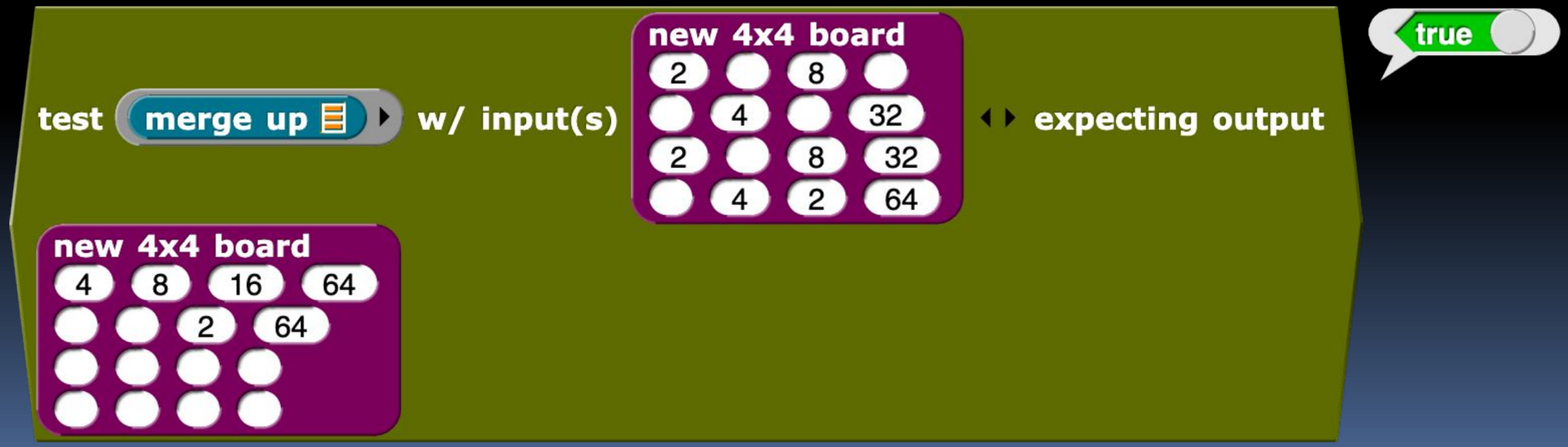
4	A	B	C	D
1	4	8	16	64
2	0	0	2	64
3	0	0	0	0
4	0	0	0	0





2048 unit tests





Summary

- Live in an immutable (functional) world if possible
- *"Testing shows the presence, not the absence of bugs"*
  – Edsger W. Dijkstra
- Proving algorithms correct requires
  - Mathematical formalism
  - Guarantee the algorithm is implemented as a function, and enumeration of all possible inputs
- Enjoy 2048!