CrowdCode: A Platform for Crowd Development

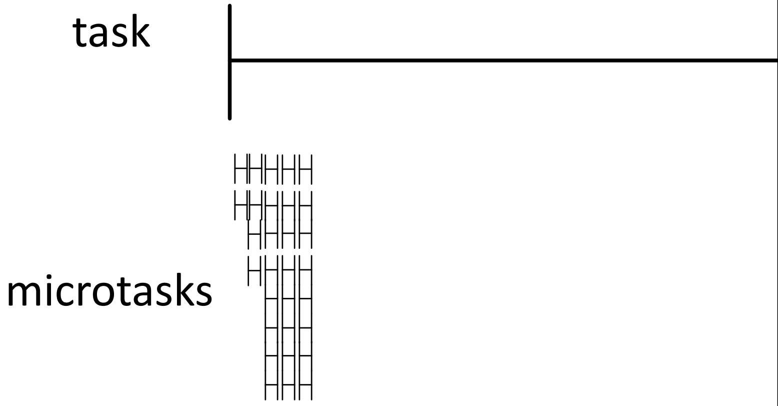
Thomas D. LaToza[1], Eric Chiquillo[1, 2], W. Ben Towne[3], Christian M. Adriano[1], André van der Hoek[1]

[1]University of California, Irvine [2] Zynga [3] Carnegie Mellon University

What if software could be built by a crowd?

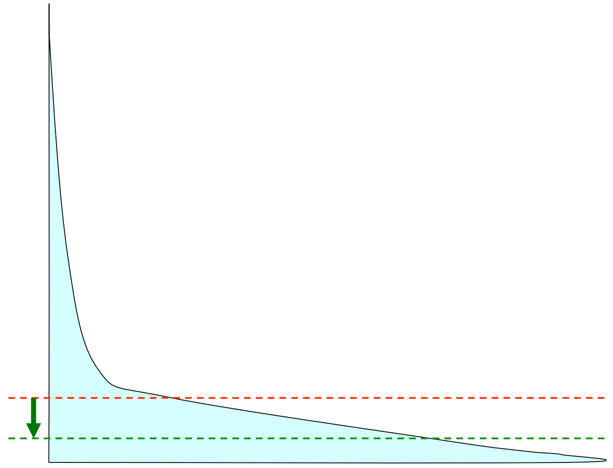
Decomposing tasks (hours - days) into microtasks (seconds - to minutes) increases **parallelism**, reducing **time** to market.

Could 1,000,000 developers build a large application in a **day**?



Lowering **joining** costs exploits the “**long tail**” of potential contributors.

Could a developer join a project, and immediately contribute?



Could this work?

Let’s find out!

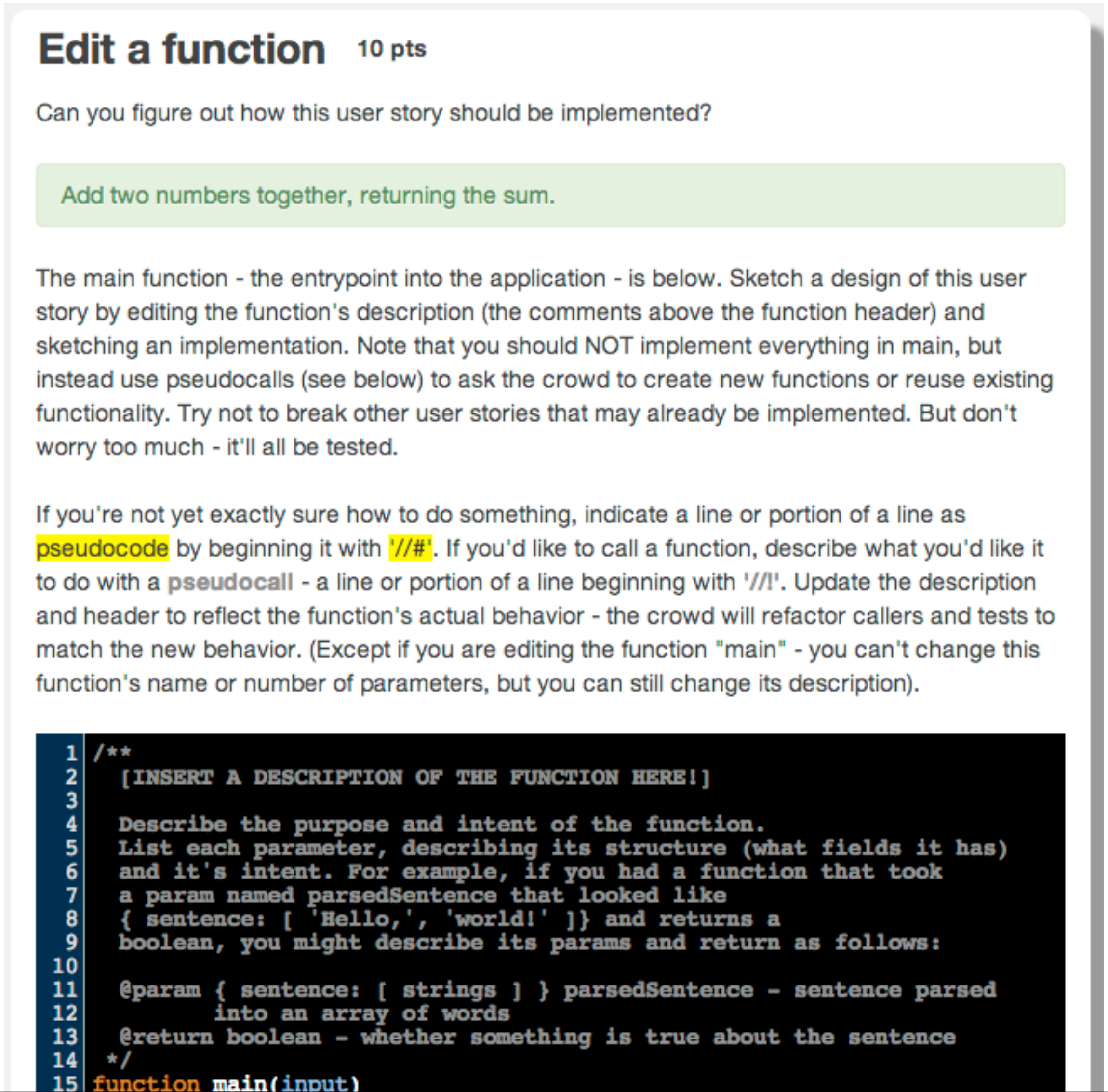
Microtasking programming

Workers work on a **single** function or test at a time, decomposing tasks to implement a feature or fix a bug into **many** microtasks that can be done in **parallel** by the crowd.

```
/**
  [INSERT A DESCRIPTION OF THE FUNCTION HERE!]

  Describe the purpose and intent of the function.
  List each parameter, describing its structure (what fields it has)
  and it's intent. For example, if you had a function that took
  a param named parsedSentence that looked like
  { sentence: [ 'Hello,', 'world!' ]} and returns a
  boolean, you might describe its params and return as follows:

  @param { sentence: [ strings ] } parsedSentence - sentence parsed
         into an array of words
  @return boolean - whether something is true about the sentence
 */
function main(input)
{
    return add(1, 2);
}
```

Self-contained microtasks

Microtasks are designed to provide **self-contained**, well-defined tasks, including all information necessary, allowing transient workers to login and immediately begin work.



Edit a function 10 pts

Can you figure out how this user story should be implemented?

Add two numbers together, returning the sum.

The main function - the entrypoint into the application - is below. Sketch a design of this user story by editing the function's description (the comments above the function header) and sketching an implementation. Note that you should NOT implement everything in main, but instead use pseudocalls (see below) to ask the crowd to create new functions or reuse existing functionality. Try not to break other user stories that may already be implemented. But don't worry too much - it'll all be tested.

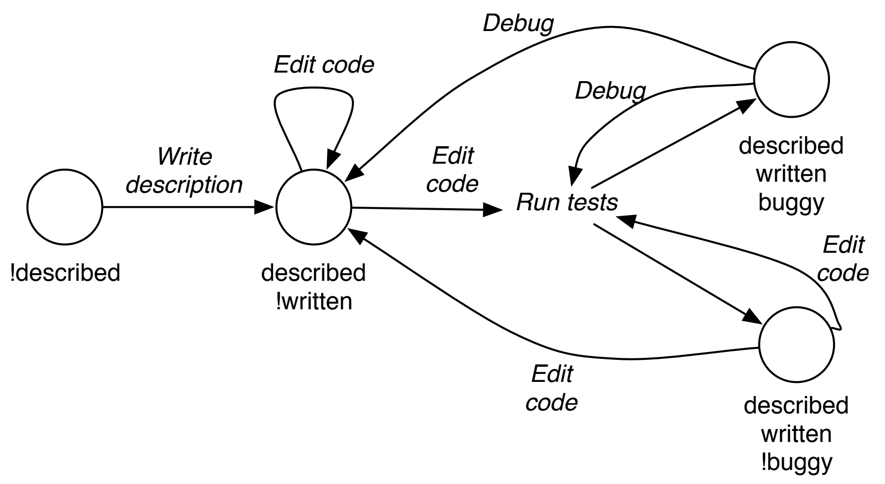
If you're not yet exactly sure how to do something, indicate a line or portion of a line as pseudocode by beginning it with '//#'. If you'd like to call a function, describe what you'd like it to do with a pseudocall - a line or portion of a line beginning with '//!'. Update the description and header to reflect the function's actual behavior - the crowd will refactor callers and tests to match the new behavior. (Except if you are editing the function "main" - you can't change this function's name or number of parameters, but you can still change its description).

```
/**
  [INSERT A DESCRIPTION OF THE FUNCTION HERE!]

  Describe the purpose and intent of the function.
  List each parameter, describing its structure (what fields it has)
  and it's intent. For example, if you had a function that took
  a param named parsedSentence that looked like
  { sentence: [ 'Hello,', 'world!' ]} and returns a
  boolean, you might describe its params and return as follows:

  @param { sentence: [ strings ] } parsedSentence - sentence parsed
         into an array of words
  @return boolean - whether something is true about the sentence
 */
function main(input)
```

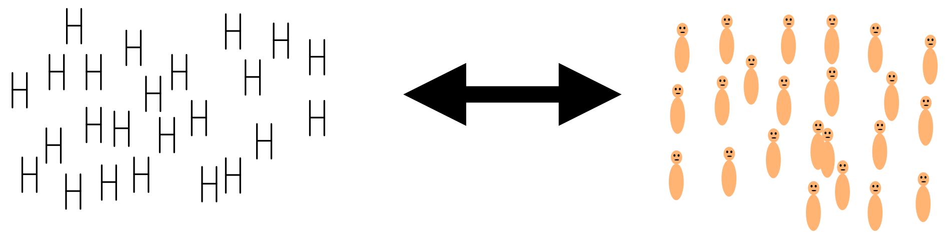
Automatic task generation

System tracks the **state** of each artifact, determine work to be done, **generates** microtasks.

Function state machine



Workers login to system and are **automatically** assigned a microtask.



Key simplifications

Work begins with a set of **user stories** (scenarios) specified by **client** which do not change.
    Each user story can be tested by a set of tests of a main() function.

Functions are completely specified by their **inputs and outputs**. (e.g., a library)
    Functions do not mutate global state or interact with environment (e.g., write output).
    All bugs can be detected through unit tests.

Programs are written in a (basic subset of) Javascript (e.g., no callbacks).

Programming tasks are to **implement** a feature, **fix** a bug, write **tests**.

All **design** is done locally and iteratively (e.g., through refactoring).

Workers are **motivated** by pay or reputation and **not malicious**.

==> crowdsourcing the programming of functional Javascript libraries

Demo!

Challenges crowdsourcing software development

- How does the system **generate** microtasks?
- How can microtasks be done **concurrently**?

How does the system generate microtasks?

Existing approaches to crowdsourcing complex work rely on a fixed sequence of steps.

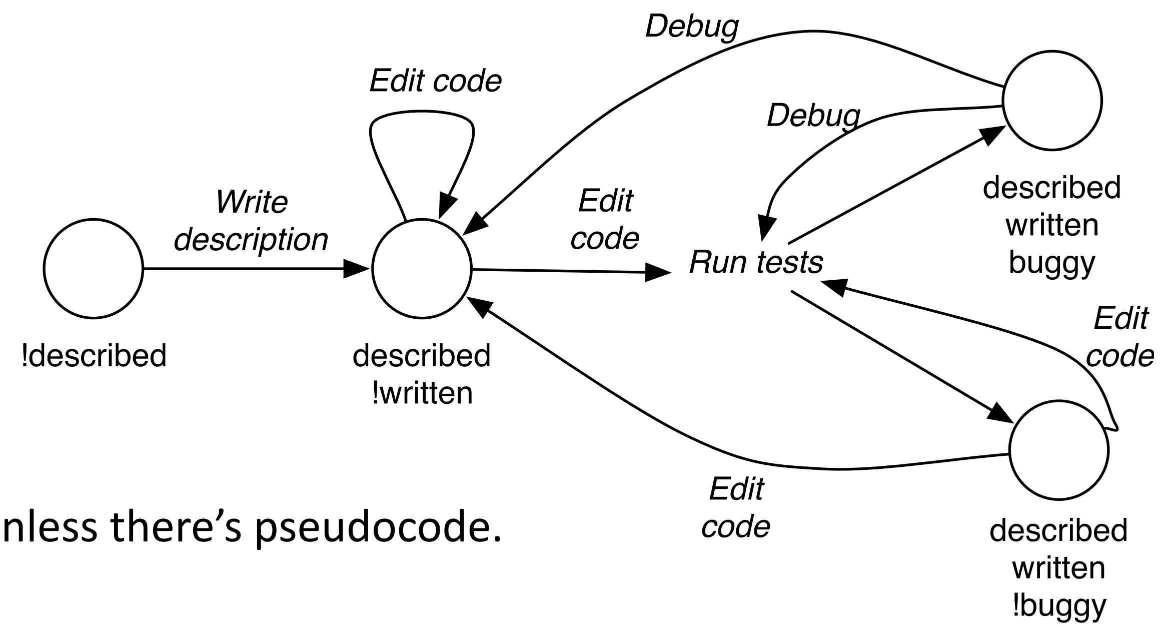
e.g., map reduce approach

Software development is dynamic, cannot enumerate tasks in advance

Each artifact has **attributes** describing its state. Submitting microtask may change a function's attributes, **generating** microtasks.

Function state machine

written (no pseudocode)
described (has a function description)
buggy (fails test)



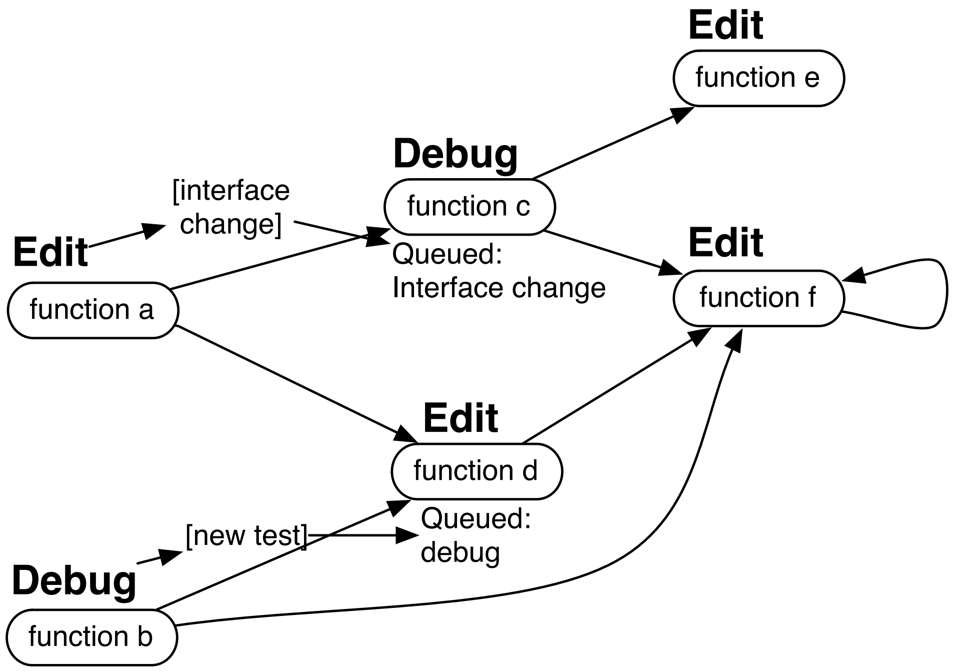
If code **changes**, run **tests**, unless there's pseudocode.

How can microtasks be done concurrently?

Software development involves **dependencies** between artifacts; as an artifact changes, others may also need to change.

All artifacts in the system may be changing concurrently.

Only **one** microtask may be active per artifact, preventing merge conflicts. Events propagate **changes** (signature, tests) across dependencies. Microtasks may "check out" a readonly copy of global interfaces, but may only commit a change to a single artifact. Events received on an artifact **queue** microtasks to be done.



Current status

Ran a pilot study, crowdsourcing a small (~500 line?) checkers program

Revealed bugs, usability issues, need for data structures.

Emergent design: aggregating conflicting local views

Multiple functions may call the same function.

What happens when their expectations **conflict**?

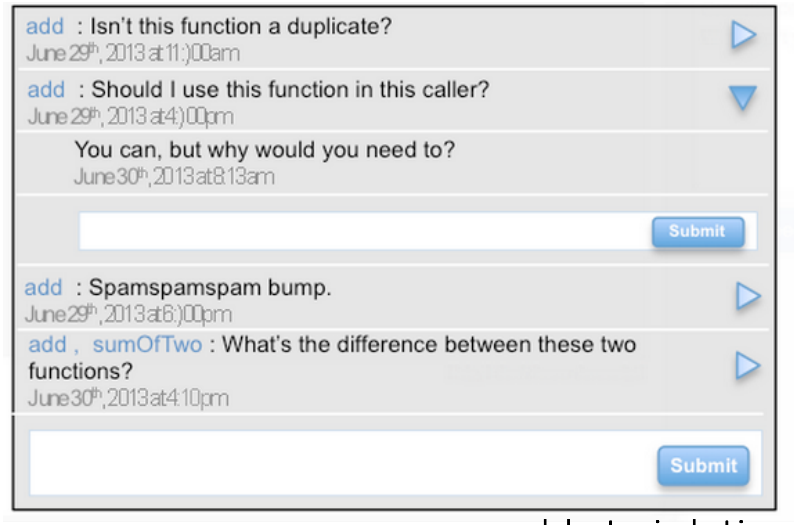
==> edit war, as functions repeatedly changed to conform to conflicting tests

Aggregation through discussion threads, bringing in relevant artifacts

Function has a discussion **thread** visible to function, tests, callers.

Workers are **transient**, so can't reference workers (e.g., they might leave).

Use @ to reference artifacts in discussion, microtask assigned to followup.



add : Isn't this function a duplicate?
June 29th, 2013 at 11:)00am

add : Should I use this function in this caller?
June 29th, 2013 at 4:)00pm

You can, but why would you need to?
June 30th, 2013 at 8:13am

add : Spamspamspam bump.
June 29th, 2013 at 6:)00pm

add , sumOfTwo : What's the difference between these two functions?
June 30th, 2013 at 4:10pm

work by Lucinda Lim

Conclusions

Programming tasks can be decomposed into microtasks at a **function** granularity.

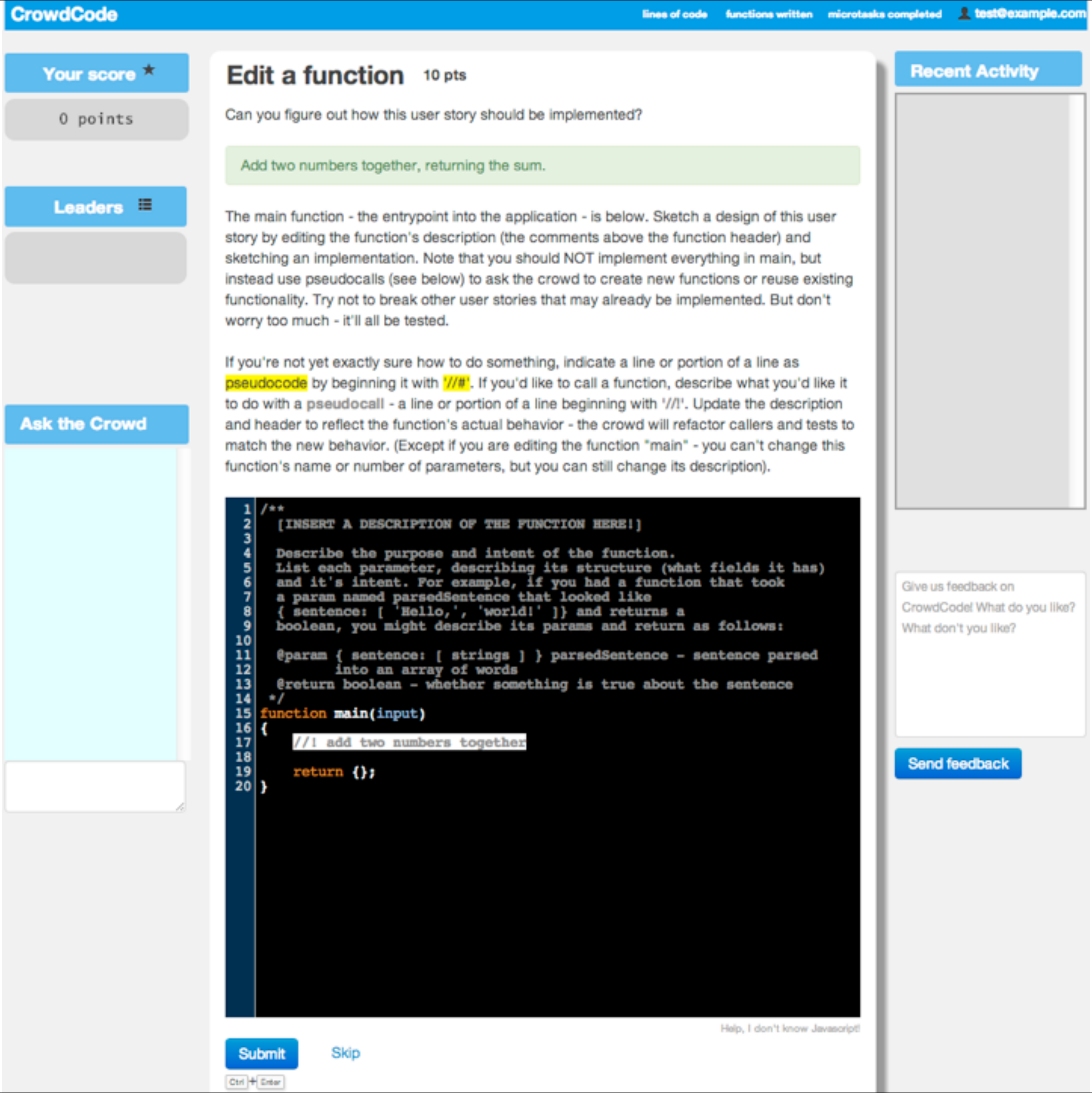
Challenges involve ensuring microtasks are **self-contained** and can be done in **parallel**.

Open questions

- Decomposition at what **granularity**?
  - Smaller -> more parallel, less entry barrier; larger -> less overhead
- How much **context** is necessary?
  - How much background about the system is necessary?
  - Tradeoffs between pulling information (Q&A) vs. pushing (reviews)
- What's the role of **design**?
  - Can software be built entirely modularly?
  - Does this increase the duplication and conflicts within the system?

Questions

Backup



Edit a function 10 pts

Can you figure out how this user story should be implemented?

Add two numbers together, returning the sum.

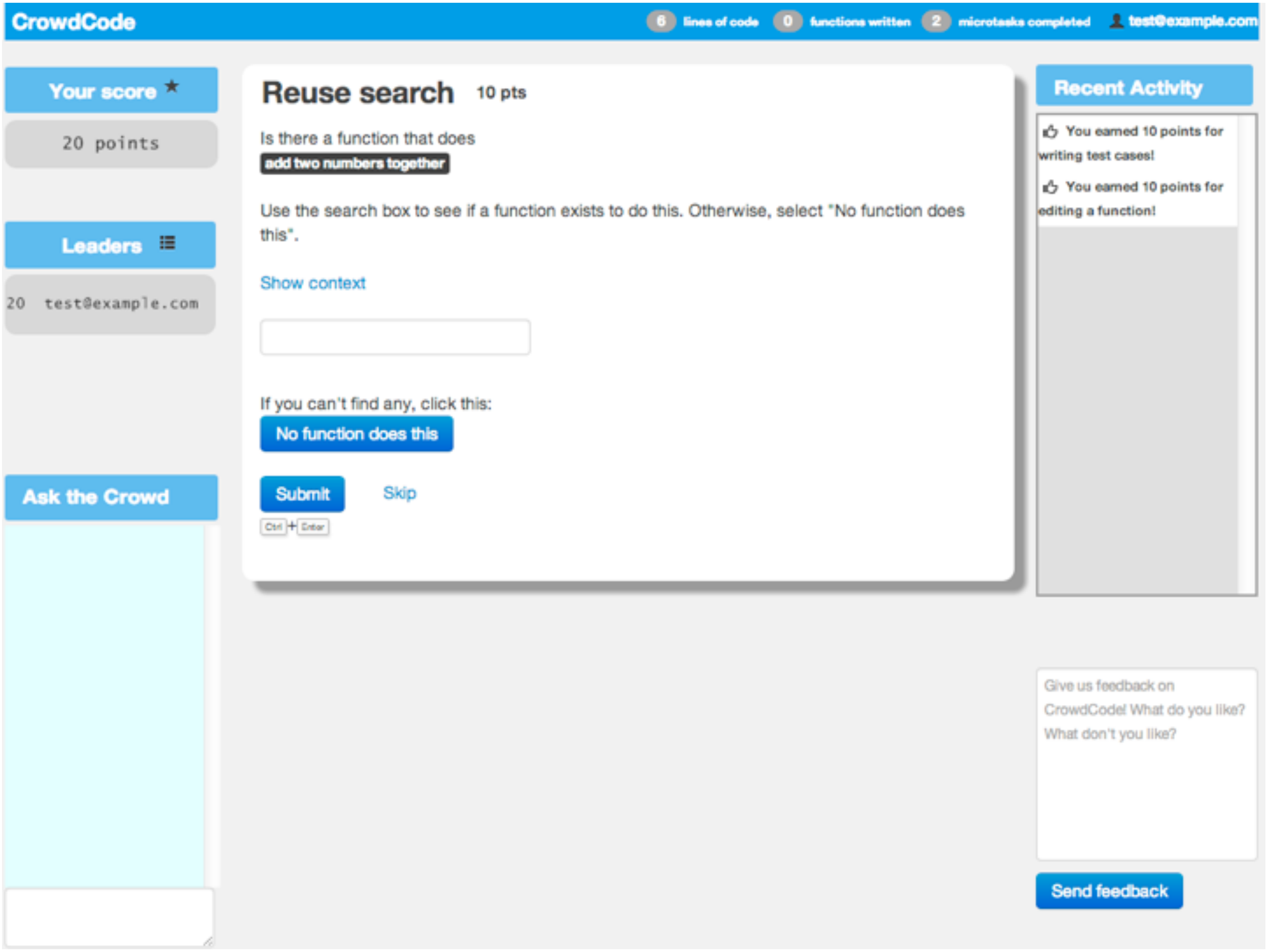
The main function - the entrypoint into the application - is below. Sketch a design of this user story by editing the function's description (the comments above the function header) and sketching an implementation. Note that you should NOT implement everything in main, but instead use pseudocalls (see below) to ask the crowd to create new functions or reuse existing functionality. Try not to break other user stories that may already be implemented. But don't worry too much - it'll all be tested.

If you're not yet exactly sure how to do something, indicate a line or portion of a line as pseudocode by beginning it with '//#'. If you'd like to call a function, describe what you'd like it to do with a pseudocall - a line or portion of a line beginning with '//!'. Update the description and header to reflect the function's actual behavior - the crowd will refactor callers and tests to match the new behavior. (Except if you are editing the function "main" - you can't change this function's name or number of parameters, but you can still change its description).

```
/**
  [INSERT A DESCRIPTION OF THE FUNCTION HERE!]

  Describe the purpose and intent of the function.
  List each parameter, describing its structure (what fields it has)
  and it's intent. For example, if you had a function that took
  a param named parsedSentence that looked like
  { sentence: [ 'Hello,', 'world!' ]} and returns a
  boolean, you might describe its params and return as follows:

  @param { sentence: [ strings ] } parsedSentence - sentence parsed
         into an array of words
  @return boolean - whether something is true about the sentence
 */
function main(input)
{
    //! add two numbers together

    return {};
}
```

Help, I don't know Javascript!

Submit
Skip
Ctrl + Enter
Give us feedback on CrowdCode! What do you like? What don't you like?
Send feedback



CrowdCode

6 lines of code 0 functions written 2 microtasks completed test@example.com

Your score

20 points

Leaders

20 test@example.com

Ask the Crowd

Reuse search 10 pts

Is there a function that does

add two numbers together

Use the search box to see if a function exists to do this. Otherwise, select "No function does this".

Show context

If you can't find any, click this:

No function does this

Submit Skip

Ctrl + Enter

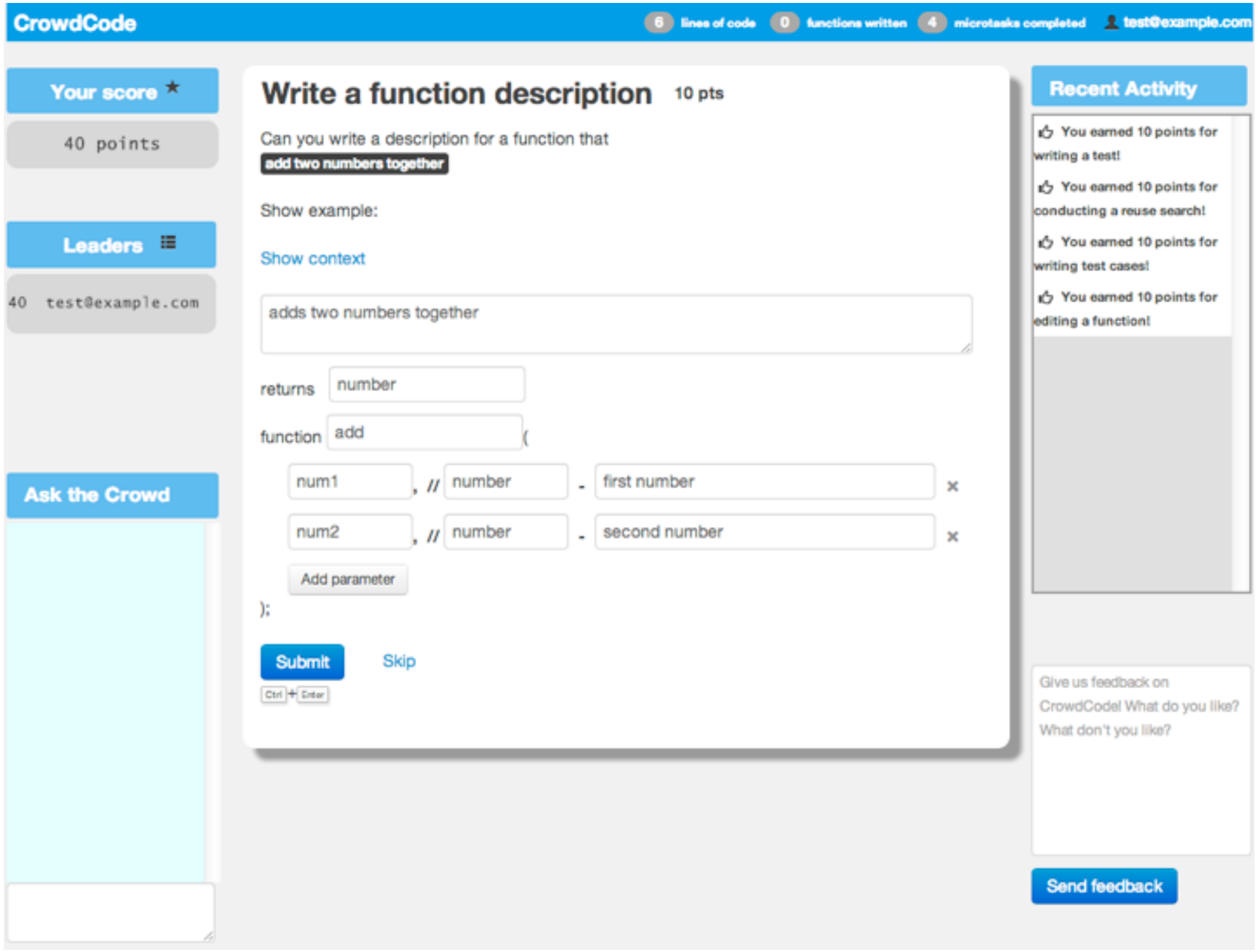
Recent Activity

You earned 10 points for writing test cases!

You earned 10 points for editing a function!

Give us feedback on CrowdCode! What do you like? What don't you like?

Send feedback



CrowdCode 6 lines of code 0 functions written 4 microtasks completed test@example.com

Your score

40 points

Leaders

40 test@example.com

Ask the Crowd

Write a function description 10 pts

Can you write a description for a function that **add two numbers together**

Show example:

Show context

adds two numbers together

returns number

function add (

num1 , // number - first number

num2 , // number - second number

Add parameter

);

Submit Skip

Ctrl + Enter

Recent Activity

You earned 10 points for writing a test!

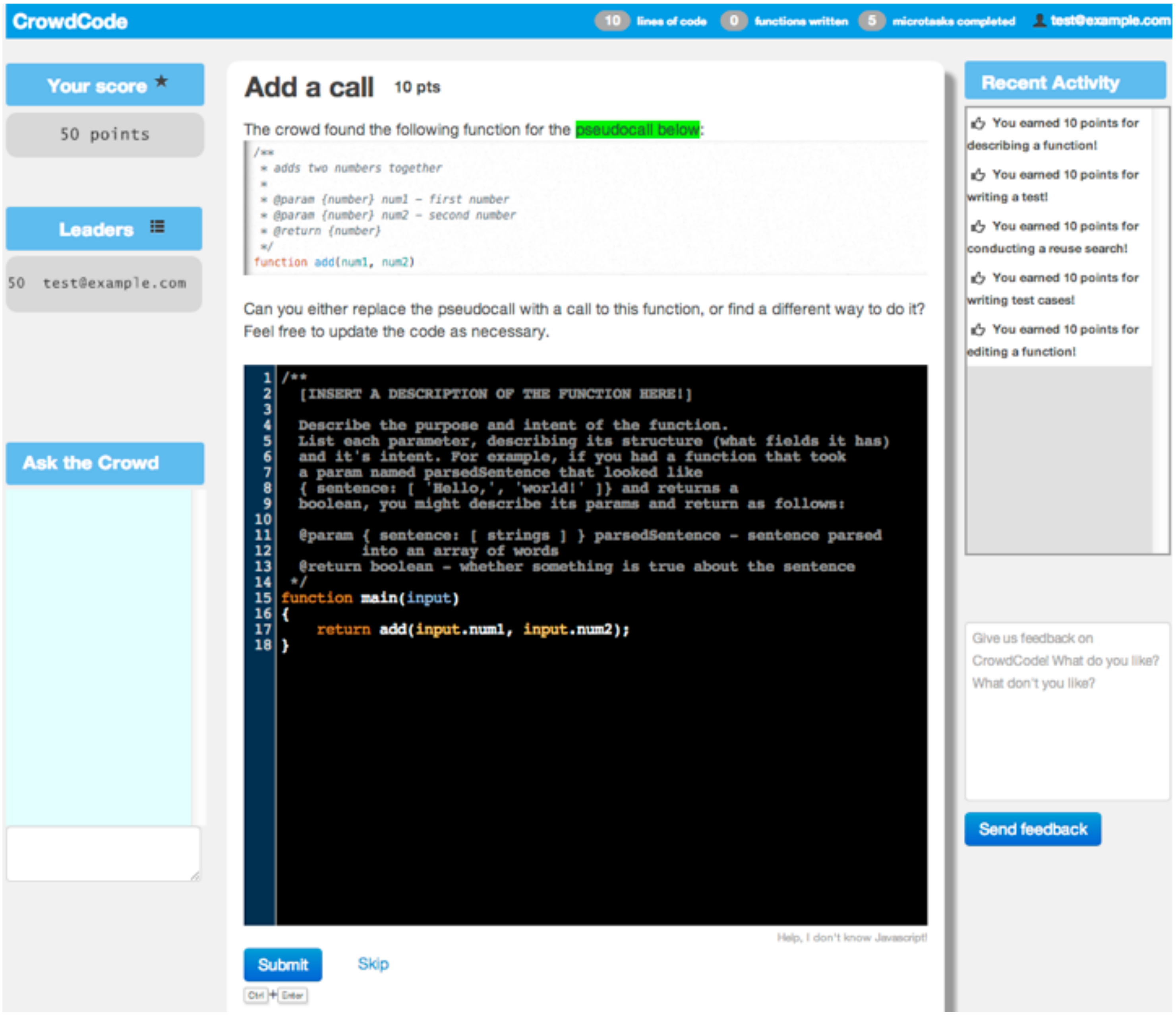
You earned 10 points for conducting a reuse search!

You earned 10 points for writing test cases!

You earned 10 points for editing a function!

Give us feedback on CrowdCode! What do you like? What don't you like?

Send feedback



Add a call 10 pts

The crowd found the following function for the pseudocall below:

```
/**
 * adds two numbers together
 *
 * @param {number} num1 - first number
 * @param {number} num2 - second number
 * @return {number}
 */
function add(num1, num2)
```

Can you either replace the pseudocall with a call to this function, or find a different way to do it? Feel free to update the code as necessary.

```
/**
   [INSERT A DESCRIPTION OF THE FUNCTION HERE!]

   Describe the purpose and intent of the function.
   List each parameter, describing its structure (what fields it has)
   and it's intent. For example, if you had a function that took
   a param named parsedSentence that looked like
   { sentence: [ 'Hello,', 'world!' ]} and returns a
   boolean, you might describe its params and return as follows:

   @param { sentence: [ strings ] } parsedSentence - sentence parsed
          into an array of words
   @return boolean - whether something is true about the sentence
 */
function main(input)
{
    return add(input.num1, input.num2);
}
```

You earned 10 points for describing a function!

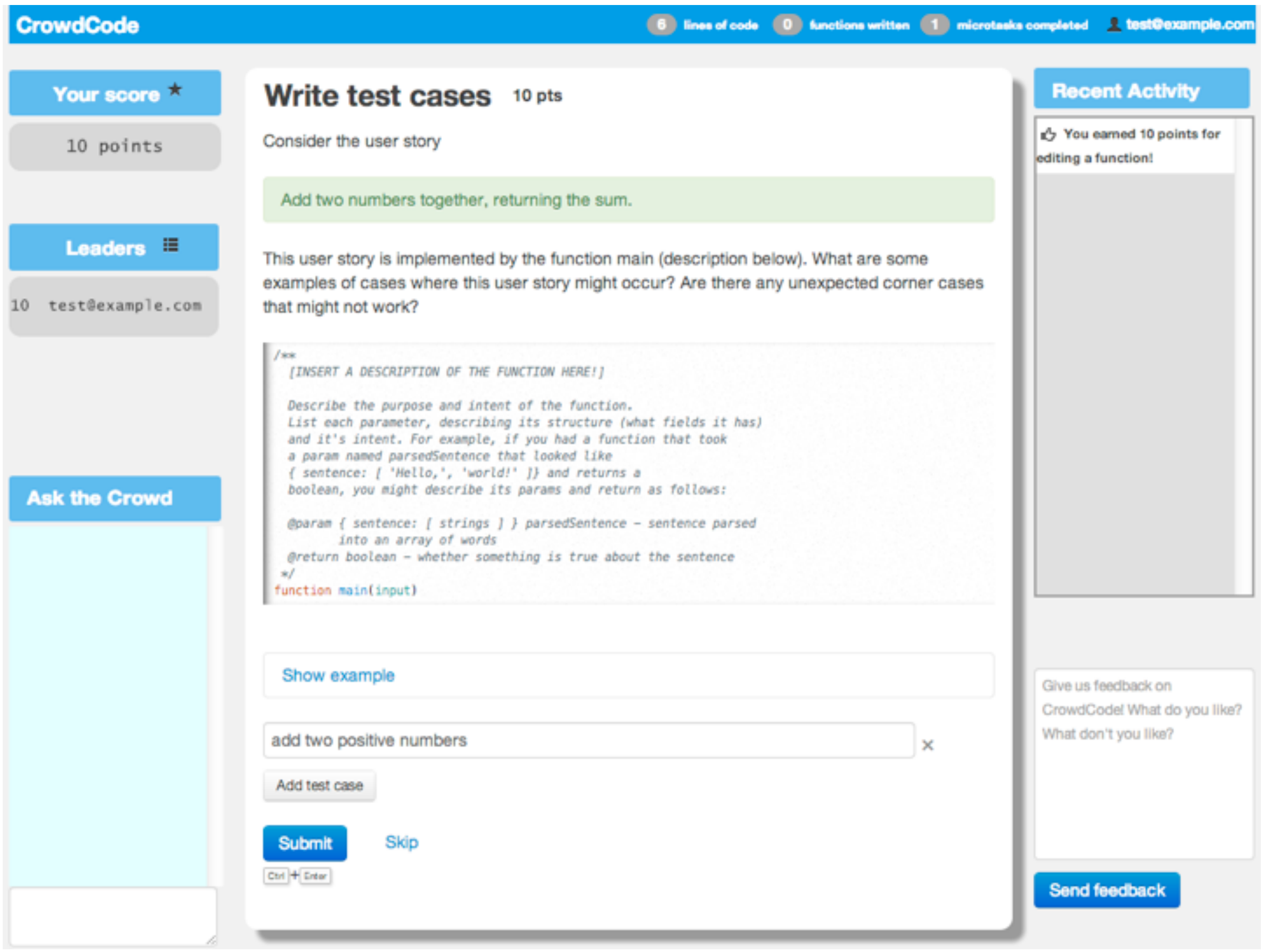
You earned 10 points for writing a test!

You earned 10 points for conducting a reuse search!

You earned 10 points for writing test cases!

You earned 10 points for editing a function!

Give us feedback on CrowdCode! What do you like? What don't you like?



10 points

10 test@example.com

Write test cases 10 pts

Consider the user story

Add two numbers together, returning the sum.

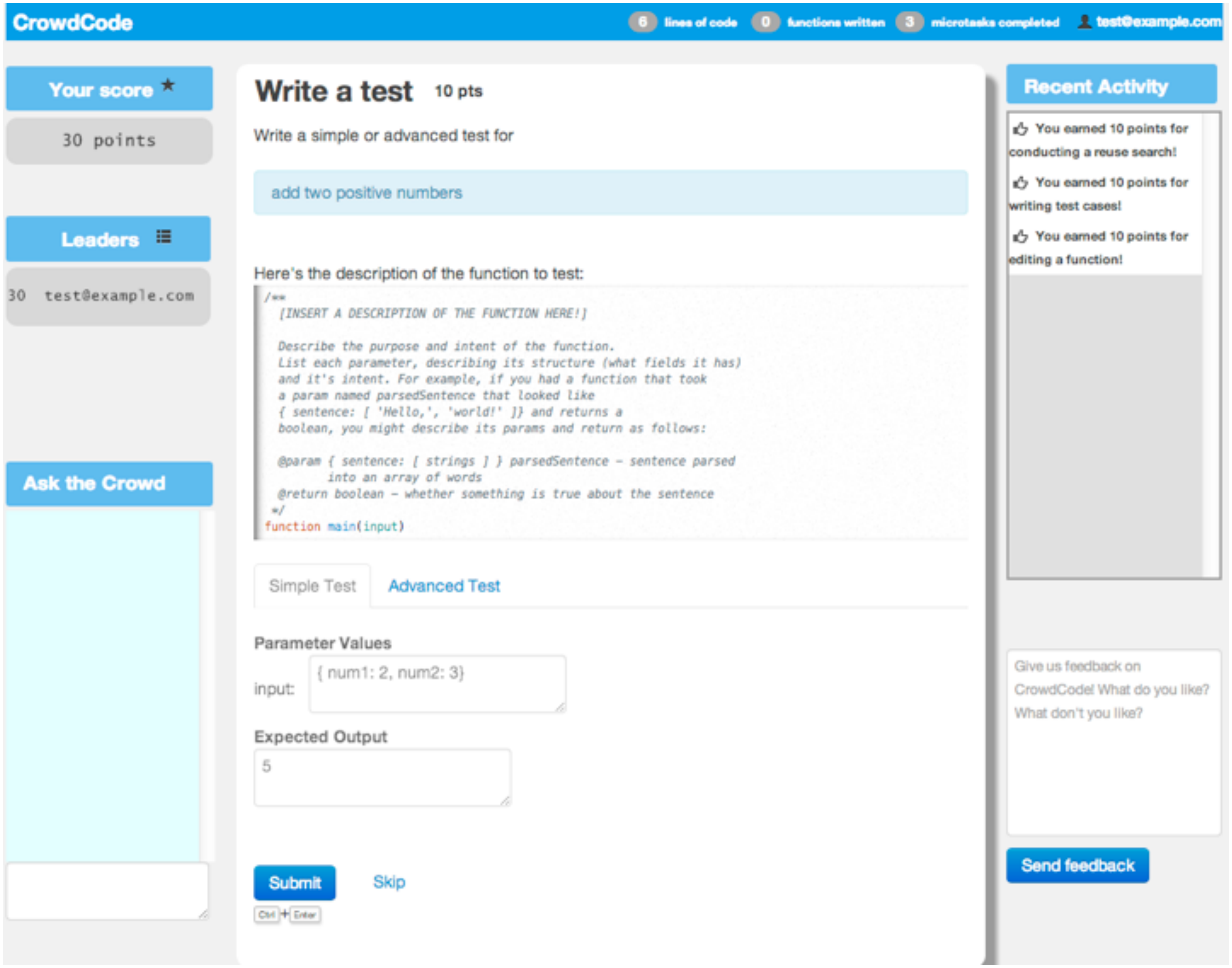
This user story is implemented by the function main (description below). What are some examples of cases where this user story might occur? Are there any unexpected corner cases that might not work?

```
/**
  [INSERT A DESCRIPTION OF THE FUNCTION HERE!]

  Describe the purpose and intent of the function.
  List each parameter, describing its structure (what fields it has)
  and it's intent. For example, if you had a function that took
  a param named parsedSentence that looked like
  { sentence: [ 'Hello,', 'world!' ]} and returns a
  boolean, you might describe its params and return as follows:

  @param { sentence: [ strings ] } parsedSentence - sentence parsed
         into an array of words
  @return boolean - whether something is true about the sentence
 */
function main(input)
```

add two positive numbers ×

You earned 10 points for editing a function!

Give us feedback on CrowdCode! What do you like? What don't you like?



CrowdCode   6 lines of code   0 functions written   3 microtasks completed   test@example.com

Write a test 10 pts

Write a simple or advanced test for

add two positive numbers

Here's the description of the function to test:

```
/**
  [INSERT A DESCRIPTION OF THE FUNCTION HERE!]

  Describe the purpose and intent of the function.
  List each parameter, describing its structure (what fields it has)
  and it's intent. For example, if you had a function that took
  a param named parsedSentence that looked like
  { sentence: [ 'Hello,', 'world!' ]} and returns a
  boolean, you might describe its params and return as follows:

  @param { sentence: [ strings ] } parsedSentence - sentence parsed
         into an array of words
  @return boolean - whether something is true about the sentence
 */
function main(input)
```

Parameter Values

input: { num1: 2, num2: 3}

Expected Output

5

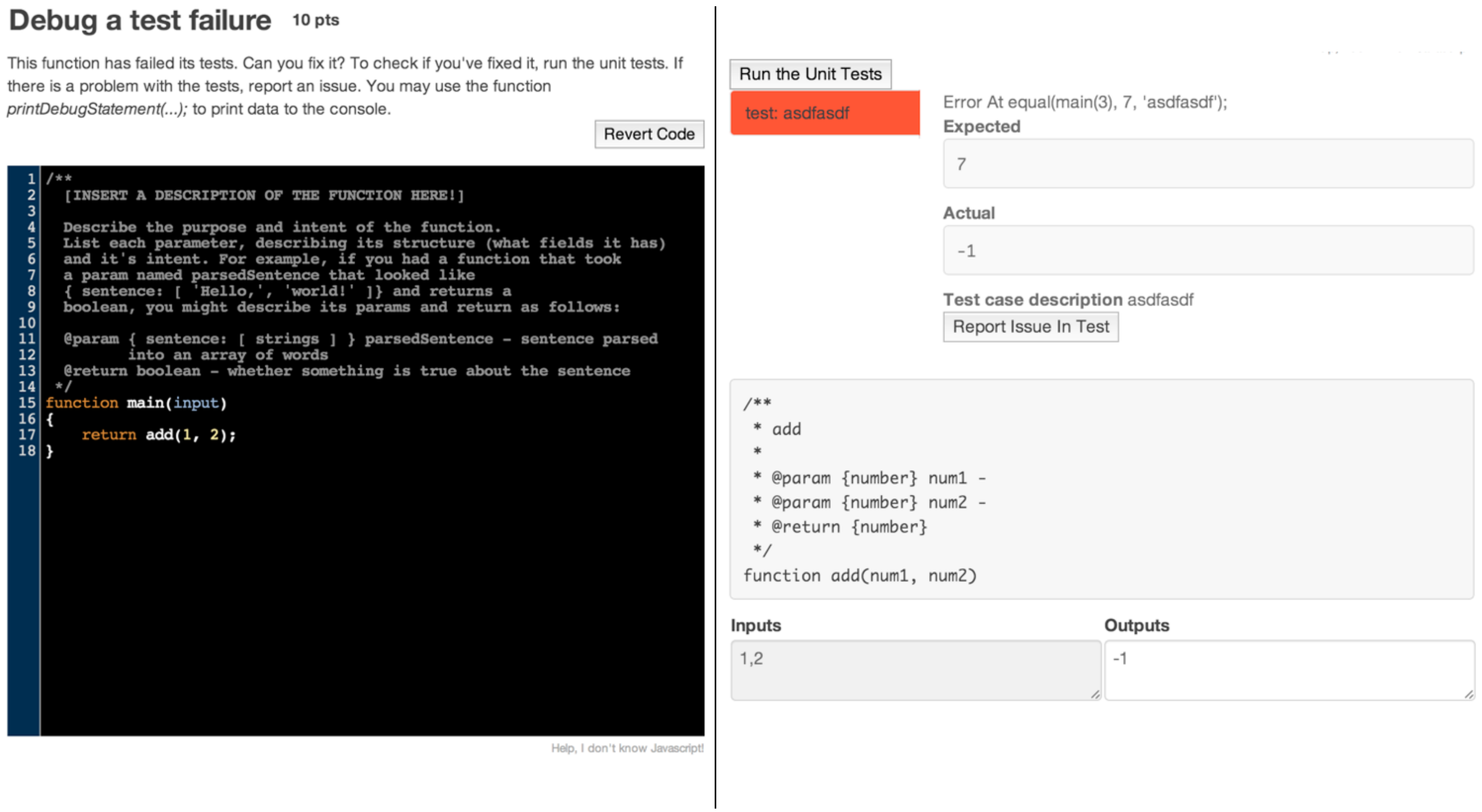
You earned 10 points for conducting a reuse search!

You earned 10 points for writing test cases!

You earned 10 points for editing a function!

Give us feedback on CrowdCode! What do you like? What don't you like?

Fixing a bug



Debug a test failure 10 pts

This function has failed its tests. Can you fix it? To check if you've fixed it, run the unit tests. If there is a problem with the tests, report an issue. You may use the function *printDebugStatement(...);* to print data to the console.

Revert Code

```
/**
  [INSERT A DESCRIPTION OF THE FUNCTION HERE!]

  Describe the purpose and intent of the function.
  List each parameter, describing its structure (what fields it has)
  and it's intent. For example, if you had a function that took
  a param named parsedSentence that looked like
  { sentence: [ 'Hello,', 'world!' ]} and returns a
  boolean, you might describe its params and return as follows:

  @param { sentence: [ strings ] } parsedSentence - sentence parsed
         into an array of words
  @return boolean - whether something is true about the sentence
 */
function main(input)
{
    return add(1, 2);
}
```

Help, I don't know Javascript!

Run the Unit Tests

test: asdfasdf

Error At equal(main(3), 7, 'asdfasdf');

Expected

7

Actual

-1

Test case description asdfasdf

Report Issue In Test

```
/**
 * add
 *
 * @param {number} num1 -
 * @param {number} num2 -
 * @return {number}
 */
function add(num1, num2)
```

Inputs	**Outputs**
1,2	-1